DM14,- Sfr14,- ÖS112,- SONDERHEFT

AMIGA

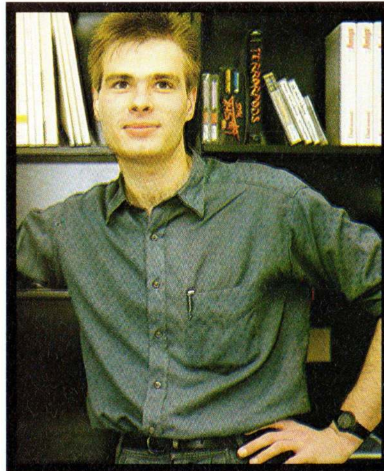
KICK START

GRAFIK SPEZIAL





Programmiergrundlagen · Viele Listings · Neue Grafiksoftware
Farbdrucker · Digitizer · Monitore · Einblicke in die Profi-Szene



Das Thema Nr.1

Nun ist es endlich soweit: Das erste Sonderheft der KICKSTART ist komplett. Natürlich lag nichts näher, als es der Sache zu widmen, zu der der Amiga nach Ansicht vieler Anwender einfach bestimmt ist: Der Grafik. Dabei haben wir uns aber, wie Sie vielleicht schon am Titelbild erkannt haben, nicht nur mit dem Amiga beschäftigt, sondern auch einmal den Profis über die Schulter geschaut. Mehr Eindrücke aus dieser Szene erwarten Sie in diesem Heft.

Dennoch dreht sich der größte Teil des Inhalts dieses Hefts natürlich um den Amiga. Für die Programmierinteressierten unter Ihnen wurde eine Auswahl an Grundlagenthemen zusammengestellt, welche die kreative Arbeit erleichtern sollen. Für diejenigen, die lieber mit fertigen Programmen arbeiten, wird eine Auswahl neuer Grafiksoftware vorgestellt. Natürlich darf die Hardware nicht fehlen, und auch diese wurde speziell unter dem Gesichtspunkt der grafischen Anwendung ausgewählt. Selbstverständlich gibt es auch eine Menge Listings; die abgedruckten Programme lassen sich in vielfältiger Weise sinnvoll anwenden und geben, wie das bei Listings so sein sollte, auch einige Beispiele für spezifische Problemlösungen, so daß zu hoffen ist, daß die Listings neben dem eigentlichen Programm einen gewissen Informationswert für Sie darstellen.

Die Grundlagen wie die Listings sind vor allem an C und Modula 2 orientiert, da nur Compilersprachen eine wirklich effiziente Programmierung ermöglichen und diese zwei dem Programmierer zur Zeit wohl die größten Möglichkeiten bieten.

Ich hoffe, daß dieses Heft viele Informationen und Anregungen für Sie mit sich bringt und wünsche Ihnen eine angenehme Lektüre.

Ihr

Wolf Dietrich

Wolf Dietrich

INHALT

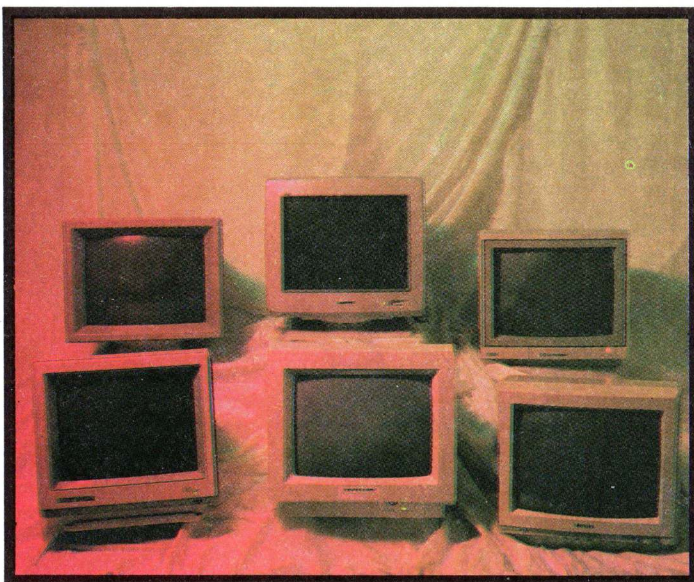
GRUNDLAGEN

SOFTWARE



Zwischen Kunst und Konstruktion
Wie arbeiten die Profis der Computergrafik-Szene? Wir präsentieren theoretische und visuelle Einblicke. Seite *10*





Der große Monitortest: Was leisten teurere Monitore im Vergleich zu preiswerteren Geräten, und für wen lohnt sich die Anschaffung eines Fremdmonitors für den Amiga? Seite **72**



Langerwartet und heißersehnt: Der Ray Tracing-Animator Animate 3D. Ein Test soll klären, ob das Programm hält, was die Ankündigungen versprochen haben. Seite **94**

HARDWARE

LISTINGS

BÜCHER

Rubrik

VON MICHAEL SISTIG

DIGITALE

BILDVERARBEITUNG

Die Stärken des Amiga liegen ja bekanntlich in seiner hohen grafischen Auflösung und der Anzahl der gleichzeitig darstellbaren Farben. Somit ist es nicht verwunderlich, daß der Amiga oft in Bereichen der digitalen Bildverarbeitung zum Einsatz kommt.

Verschiedene Hard- und Softwarehersteller bieten hier eine ganze Palette von Erweiterungen an, die diese Möglichkeiten des Amiga voll ausnutzen. PIXmate und Butcher sind zwei Programme, die speziell die Nachbearbeitungen von schon erstell-ten Grafiken ermöglichen. Ihre Stärke liegt darin, sogenannte "digitalisierte Grafiken", also Realbilder, die zuvor mit einer Videokamera in den Speicher eingelesen wurden, zu verarbeiten und zu verändern. Beide Programme verfügen hier über eine große Anzahl von Funktionen, die den Umgang mit sol-chen Grafiken zu einem wahren Genuß werden lassen. Die Software Butcher wurde von dem amerikanischen Softwarehaus "Eagle Tree Software" geschrieben. Die vorliegende Version Butcher 2.0 PAL wird mit einem 50-seitigen deutschen Handbuch ausgeliefert und unterstützt die höhere Auflösung der europäischen Amigas. Butcher ist ohne Einschränkung auf allen Amiga-Modellen (A500, A1000 und A2000) lauffähig.

Wie bei allen Grafikprogrammen ist eine Speichererweiterung nicht unangebracht, aber nicht unbedingt notwendig. Nach dem Booten der Diskette meldet sich der gewohnte Workbenchbildschirm. Butcher ist nicht kopiergeschützt, es lassen sich also Sicherungskopien oder Installationen auf Harddisk problemlos durchführen. Butcher wird durch einen Doppelklick nach dem Öffnen des Diskettenicons gestartet. Nach kurzer Ladezeit meldet sich Butcher dann zum Dienst. Hierbei fällt nach den ersten Blicken sehr positiv auf, daß nicht nur das Handbuch ins Deutsche übersetzt wurde, sondern auch alle Menüs und Requester. Ein Umstand, der Anwendern, die sich des Englischen nicht so sehr begeistern, den Umgang mit Butcher bestimmt sehr erleichtern wird. Wie alle mausorientierten Programme präsentiert sich Butcher mit verschiedenen Hauptmenüs. Diese Menüs unterteilen sich in folgende Unterpunkte:

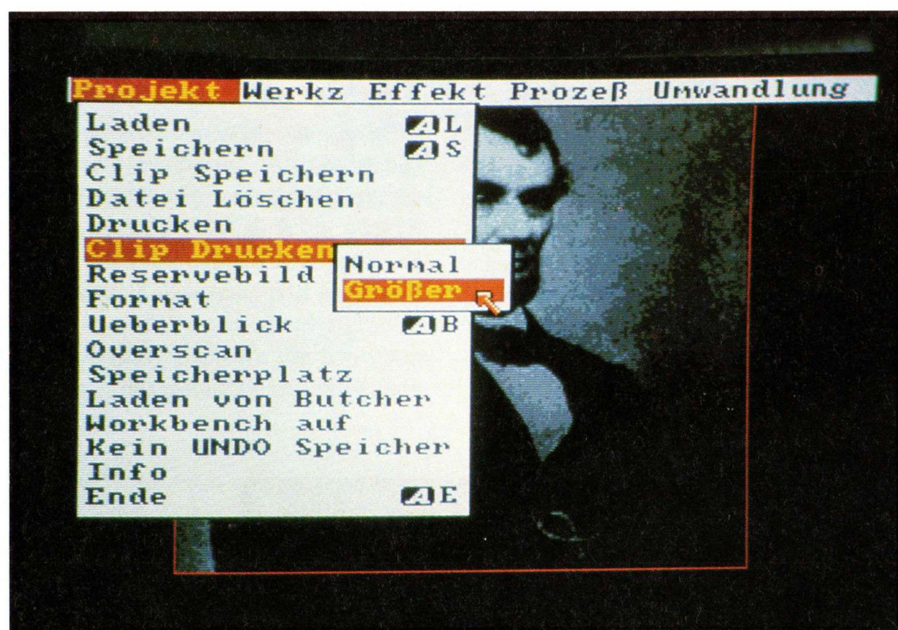
1. Projekt-Menü:

Hier findet man wie gewohnt alle Dateifunktionen wie z.B. Laden, Speichern und Löschen von Dateien. Weiterhin enthält es Funktionen wie Drucken von Bildern, Eröffnen eines zweiten Bildschirms, einen Screenformatbefehl, mit dem sich alle Bildschirmformate schnell und bequem einstellen lassen, und eine Anzahl von Befehlen für Amigabesitzer, die "nur" 512 KB Speicher haben. So kann man Butcher z.B. mitteilen, ob das gesamte Programm geladen werden soll oder nur die gerade benötigten Teile. Auch ist es möglich, den Workbenchscreen abzuschalten und so etwa 30Kbyte "Chipmem" zu sparen. Selbst der Befehl "UNDO" ist ausschaltbar, um auch hier einige Bytes zu gewinnen. Der letzte Befehl des Projektmenüs heißt "ENDE" und dient (was auch sonst ?!) zum Beenden des Programms.

2. Werkzeug-Menü:

Wie der Name schon erkennen läßt, befinden sich im diesen Menü Hilfsmittel, wie z.B "UNDO", also Rückgängigmachen von versehentlich angewählten oder falsch benutzten Befehlen. Als nützliches Tool erwies sich auch "Zeichnen". Obwohl Butcher kein Malprogramm ist, wurden hier einfache Grafikbefehle wie Linien, Rechtecke, Airbrush, Füllen etc. integriert, um schnell simple Grafiken zu erstellen. Mit "CLIP" ist es möglich, beliebige Teile eines Bildes einzurahmen und somit praktisch auszuschneiden.

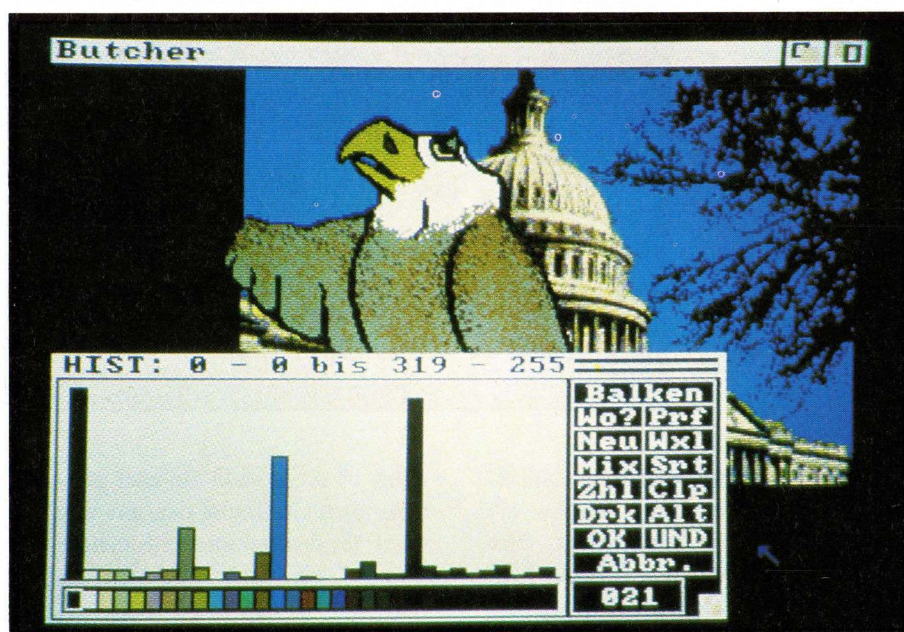
Wurde ein "CLIP" definiert, wirken sich alle Befehle von Butcher auf diesen Bildteil aus. Es ist auch möglich, einen geclippten Bildteil zu speichern oder ausdrucken zu lassen. Um wieder das ganze Bild bearbeiten zu können, wählt man "CLIP Leer".



Sämtliche Menüs von Butcher wurden ins Deutsche übersetzt.

"Bild Löschen" löscht den momentanen Bildschirminhalt und kann jederzeit mit den "UNDO"-Befehl rückgängig gemacht werden. Mit "Spiegeln" und "Kippen" läßt sich der Bildschirminhalt entweder horizontal oder vertikal verändern. Der Befehl "Zyklus" ist bestimmt schon einigen Lesern oder Besitzern von Dpaint bekannt. Er startet den zyklischen Farbverlauf, d.h die Farben laufen über einen vorher eingestellten Bereich, oder Teilbereich der Farbpalette, durch. Hier lassen sehr schöne Animationseffekte erzielen, die sich auf vier Teilbereiche der aktuellen Palette auswirken und unterschiedliche Farbscrollgeschwindigkeiten haben können. Mit den Befehlen "Farbpaletten" und "Neue Palette" lassen sich eigene Farbpaletten erstellen. Butcher erlaubt es, bis zu 3 unterschiedliche Paletten zu benutzen. Hat man einen dieser zwei Befehle angewählt, erscheint ein Requester (siehe Bild 1), mit dem man sich seine individuelle Palette mischen kann. Man kann die Farben mit Hilfe eines RGB- (Rot-Grün-Blau) oder TSW-Reglers (Farbton- Farbsättigung- Farbwert) mischen. Farben können nach ihrer Intensität sortiert werden und lassen sich mit Befehlen wie "NEG" (dunkle Farben werden hell und umgekehrt) t oder mit "KMP" (bildet die Komplementärfarbe) beliebig verändern.

Weiterhin bietet der Farbrequester die Möglichkeit, Farbverläufe zwischen einer beliebigen Start- und Endfarbe zu bilden. Mit "MAP" lassen sich dann einmal gestaltete Paletten auf andere Paletten übertragen. Der Requester beinhaltet noch eine ganze Anzahl von weiteren Farbmanipulationsmöglichkeiten, die aber den Umfang dieses Testberichts sprengen würden.



Mittels eines Histogramms lassen sich sämtliche Farbwerte exakt anzeigen.

3. Die Effekte-, Prozess- und Umwandlungmenüs

Diese Menüs bilden das eigentliche Kernstück von Butcher. Hier befinden sich alle Funktionen, die sich direkt auf die eigentliche Bildmanipulation beziehen. So ist es z.B möglich, jedes geladene Bild mit den verschiedensten Befehlen nachzubearbeiten. Man kann Bilder (jeder Auflösung!) miteinander kombinieren, vermischen und überlagern. Im Effektemenü verbergen sich die direkten Farbveränderungsbefehle. Hier ist es möglich, Bilder zu tönen, Farben einzeln zu trennen, Komplemente zu bilden, Farbbilder in Schwarzweißbilder umzuwandeln oder auch Pseudofarben zu erstellen (d.h. die Farben in Abhängigkeit ihrer Intensität zu verändern). Man findet hier auch einen Effekt, der "Antik" heißt. Mit ihm ist es möglich, digitalisierte S/W-Bilder antik erscheinen zu lassen. Der Rechner ändert dann die Farben so um, daß man glaubt, eine alte,verbleichte S/W-Fotografie vor sich zu sehen. Auch hier ist es unmöglich, die Vielzahl der Funktionen zu erklären, Abhilfe schafft hier nur das genaue Studium des zum Glück deutschen Handbuchs. Im Menü "Prozess" findet man sehr nützliche Befehle, um Konturen oder harte Farbübergänge zu

verwischen. So ist es möglich, mit Befehlen wie z.B "Ränder" Konturen zu erkennen und zu beeinflussen. Hierbei ist es möglich, dem Programm mitzuteilen, in welchen Farbbereich die Kontur gesucht werden soll, und wie stark sich Farbübergänge bei der Suche auswirken sollen. Somit ist es möglich, sich zum Beispiel nur alle roten Konturumrisse eines Farbbildes aus dem Bild lösen zu lassen. So kann man schnell, und ohne viel Mühe, von aus einen 32farbigen Low-Res-Bild (5 Bitplanes) ein 2farbiges (1 Bitplane) Bild erzeugen. Auch besteht die Möglichkeit, Bilder, die im Hold-and-Modify-Modus (HAM-Mode, 4096 Farben) vorliegen, in normale Low-Res-Bilder zu konvertieren. Dies erweist sich gerade beim Bearbeiten digitaliserter Bilder als nützlich, um diese z.B. mit Dpaint weiterbearbeiten zu können.

Der Intro - Screen von PIXmate



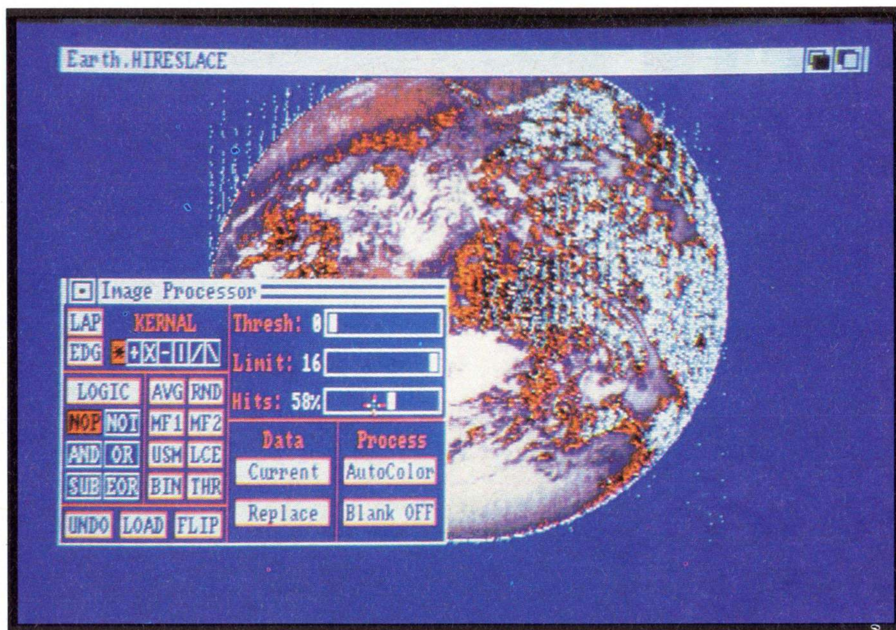
digitalisierten Fotos oder anderen Grafiken Silhouetten oder andere interessante Motive herauslösen. Mit "Rasterung" hat man eine schnelle Methode, Bilder in Graustufen darstellen zu lassen. Masken zur Rasterung und Intensität sind hierbei frei wählbar. Um einen genauen Überblick über die Anzahlen der verwendeten Farben des jeweiligen Bildes zu erhalten, kann man unter dem Menü "Umwandlung" z.B. den Befehl "Histogramm" wählen. Hier erhält man eine statistische Übersicht in Form eines Balkendiagramms, die die Häufigkeit der verwendeten Farben anzeigt. Ersieht man hieraus, daß bestimmte Farbregister, die man vielleicht benutzen will, belegt sind, kann man mit "Konsolidierung" ähnliche Farben auf ein Register zusammenfassen, und sich so Platz für weitere Farben schaffen. Mit "Bit-Tiefe" schaltet man die einzelnen Bitplanes des Amiga aus. So kann man

Alles in allem stellt Butcher ein sehr luxuriöses Grafiktool dar, das sich sowohl für den semiprofessionellen Anwender wie auch für den eingefleischten Amiga-Grafikfreak zu erstehen lohnt. Meiner Meinung nach müßte das Handbuch noch einmal überarbeitet werden (daß es in Deutsch ist, ist ja gut, aber zu viele Funktionen des Programms werden nicht genau erklärt, so daß man einige Zeit braucht, um mit Butcher die Ergebnisse zu erzielen, die das Programm hergeben kann). Um die Geschwindigkeit einiger Grafikroutinen sollten sich die Programmierer noch einmal kümmern, hier könnte noch einiges optimiert werden.

Die Konkurrenz: PIXmate

PIXmate ist ein neues Programm auf dem Markt, das, laut seinem Hersteller, der amerikanischen Firma Progressive Peripherals & Software, einiges mehr als vergleichbare Programme (wie Butcher) hergeben soll. So wird mit selbstgeschriebenen "Blitter"-Routinen und mit mehr als 3000 (!) Bildmanipulationsmöglichkeiten geworben. Solche Schlagworte klingen vielversprechend, und uns interessierte, was dahintersteckt.

Im Lieferumfang befinden sich ein ca. 180seitiges, sehr ausführliches englisches Handbuch und die Programmdiskette mit PIXmate. Das Programm ist wie Butcher nicht kopiergeschützt, und läuft auch auf allen Amigasystemen. Um alle Möglichkeiten voll ausschöpfen zu können, empfiehlt der Hersteller einen Speicherausbau um mindestens 1 Megabyte. Trotzdem ist PIXmate auch mit "nur" 512 K voll einsatzfähig, weil es auch hier wie bei Butcher die Option gibt, nur dann bestimmte Teile nachzuladen, wenn sie vom Programm benötigt werden. Hat man trotzdem zu wenig Platz, kann man, analog zu Butcher, wertvolle Bytes durch Ausschalten speicherintensiver Funktionen gewinnen. Um PIXmate zu starten, muß man die Workbench booten und



Der Image-Prozessor von PIXmate bietet über 3000 Kombinationsmöglichkeiten.

PIXmate mit einem Doppelklick aktivieren. Nach einem beeindruckenden Titelbild ist PIXmate dann bereit, seinen Dienst anzutreten.

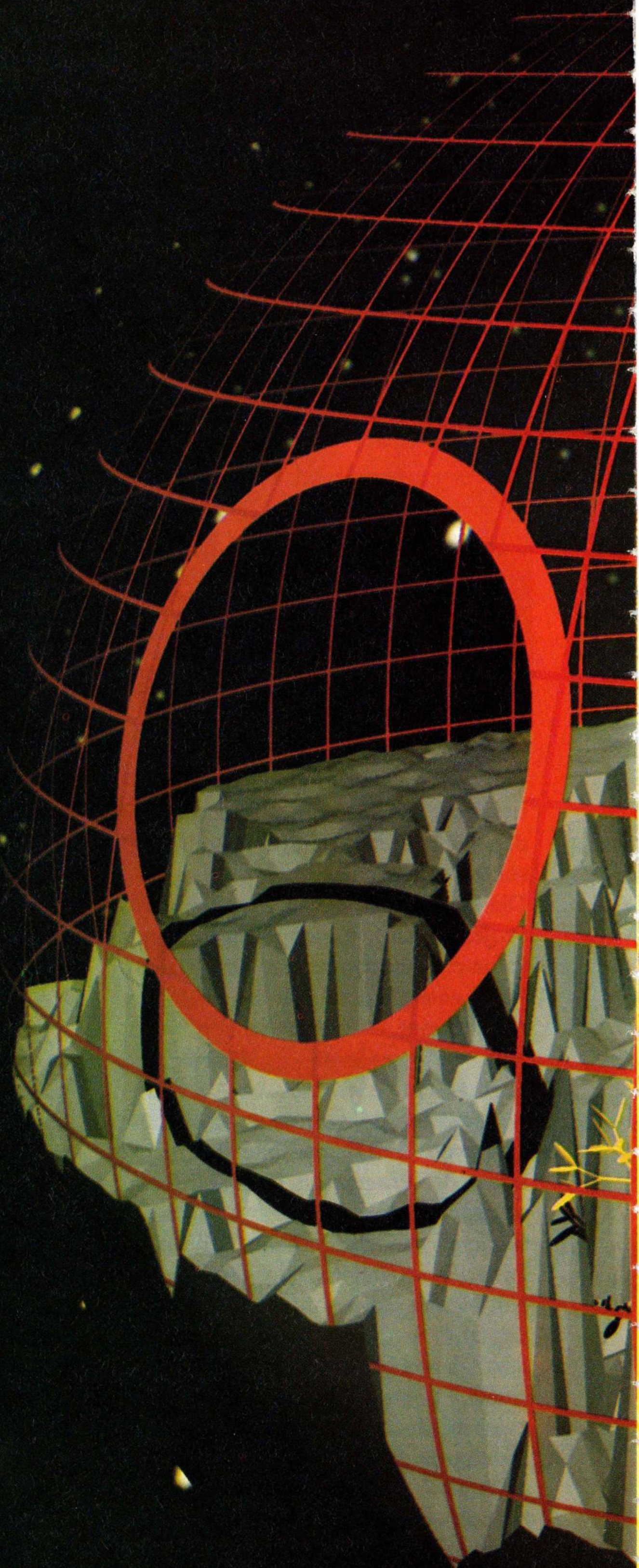
Die Funktionen von PIXmate sind im Project-, Edit-, Color-, Effects- und Infomenü untergebracht. Das Handbuch untergliedert sich in zwei Bereiche: eine "Tutorial-Section" und eine sogenannte "Reference-Section". In ersterwähnten Abschnitt werden, anhand von leicht nachvollziehbaren Beispielen, alle Möglichkeiten des Programms kurz umrissen, so daß man den Umgang mit PIXmate spielend erlernen kann. Witzige Passagen im Handbuch lockern die Arbeit hierbei auf, und die erstklassigen digitalisierten Demo-Grafiken lassen das Ausprobieren des Programms zu einem wahren Spaß werden. Im zweiten Abschnitt des wirklich guten Handbuchs findet man dann alle Befehle von PIXmate ausführlich behandelt und erklärt. Im direkten Vergleich zu Butcher kann man sagen, daß alle Funktionen, die man von Butcher her kennt, nicht fehlen. Auffällig ist die hohe Verarbeitungsgeschwindigkeit - die auf das geschickte Programmieren von eigenen Blitterroutinen hindeutet-, mit denen das Umrechnen der Bilder geschieht. Komplexe Berechnungen wie z.B das logische Verknüpfen von einzelnen Pixeln laufen um den Faktor 3 bis 5 schneller als bei Butcher.

Auch wartet PIXmate bei der Anzahl der Bildmanipulationsmöglichkeiten mit mehr Befehlen auf. Diese werden im Menü "Effects" mit dem Befehl "Image Process" aktiviert. Hier erscheint dann ein Requester, mit dem sich alle direkt auf das Bild wirkenden Operationen steuern lassen. So man hat die Auswahl zwischen verschiedenen logischen (wie AND, OR, XOR ... etc) und pixelorientierten Operationen, die man beliebig miteinander kombinieren kann. Alleine die hieraus resultierenden Möglichkeiten können einen interessierten Anwender Monate vor dem Bilschirm fesseln. Ein weiteres Plus von PIXmate ist sein ausgeklügeltes FileHandling-System. So ist es möglich, Bilder, die mit dem ATARI ST und den Programmen NEOCHROM oder DEGAS erstellt wurden, in den Amiga zu laden. Auch lassen sich selbsterstellte oder veränderte Grafiken nicht nur im IFF-Format abspeichern, sondern auch in einem PIXmate-spezifischen "RAW-FORMAT". Da diese Grafiken um 30-50% weniger Platz benötigen, läßt sich der Speicherplatz der Disk um fast die Hälfte vergrößern. Mit "Save Palette" läßt sich jede Farb-Palette, ob selbst erstellt oder übernommen von einer geladenen Grafik, seperat auf die Disk speichern. Solche Paletten lassen sich dann jederzeit wieder in den Speicher zurückholen und mit Paletten geladener Bilder logisch verknüpfen. Vielen Leuten, aber scheinbar nur wenigen Programmierern, ist bekannt, daß der Amiga über "Multitasking" verfügt, sprich die Möglichkeit, mehrere Programme gleichzeitig bearbeiten zu können. Nicht so den Programmierern von PIXmate. Das Programm bietet die Möglichkeit, z.B. DeLUXE Paint II oder ein anderes Malprogramm im Hintergrund laufen zu lassen (Voraussetzung ist hier eine Speicherweiterung um mindestens 1 MegaByte !) und sich jederzeit den aktuellen Bildschirm in PIXmate zu holen und dort weiterzuverarbeiten. Dies ermöglicht es, einen direkten Austausch von Daten vorzunehmen, der den Anwender bei anderen Programmen leicht zum Diskjockey werden lassen kann. Die Liste der Funktionen im Vergleich zu anderen Programmen, wie z.B. Butcher, ließe sich beliebig erweitern, würde aber den Rahmen dieses Berichts sprengen. Bleibt zu sagen, daß Butcher und PIXmate wohl zur Zeit die besten Grafikprogramme im Bildmanipulationsbereich am Amiga-Softwarehimmel darstellen, obwohl ich PIXmate den Vorrang in Bezug auf seine Geschwindigkeit und Erlernbarkeit geben würde. Nachteilig erachte ich bei PIxmate eigentlich nur, daß das wirklich gutgeschriebene Handbuch leider in Englisch gehalten ist. Trotzdem kann man beide Programme wirklich jedem grafikinteressierten Anwender voll und ganz empfehlen.

Butcher 2.0 PAL
Herst.: Eagle Tree Software
Preis: 115.- DM
Vertrieb: PDC, Bad Homburg, 06172/ 24748
PIXmate
Herst.: Progressive Periphals & Software
Preis: 149.- DM
Vertrieb: IM, Frankfurt, 069/7071102

ZWISCHEN KUNST UND KONSTRUKTION

Faszination der Computergrafik

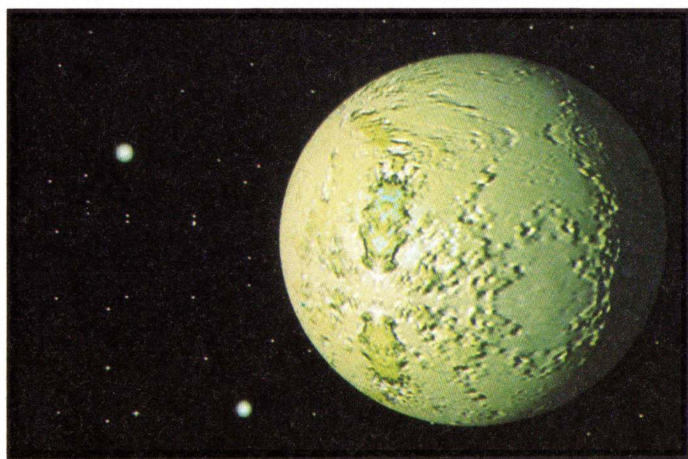


VON CHRISTIAN SCHORMANN

Die Anwendung von Computern für grafische Aufgaben hat in den letzten Jahren stark zugenommen, einhergehend mit den fallenden Preisen für leistungsstarke Computer, wie auch mit gesteigertem Know How, das an zahlreichen Forschungsstätten vor allem in Amerika und Japan erworben wurde.

Die Spanne der professionellen Anwendungen beginnt mit Malprogrammen. So etwas kennt wohl jeder unserer Leser von seinem Amiga. Professionelle Systeme können allerdings etwas mehr, beziehungsweise in besserer Qualität. Verwendung finden solche Systeme vor allem im Broadcastbereich, also bei der Bearbeitung von Fernsehbildern, und in der Grafikerbranche.

Ein anderer Typ von Systemen dient als Konstruktionshilfe; der Computer wird als extrem komfortables Zeichenwerkzeug mit Anwendungen von der Architektur bis zum Maschinenbau benutzt.



Copyright '87, by Art Com Atelier für Computergrafik GmbH, Bremen.

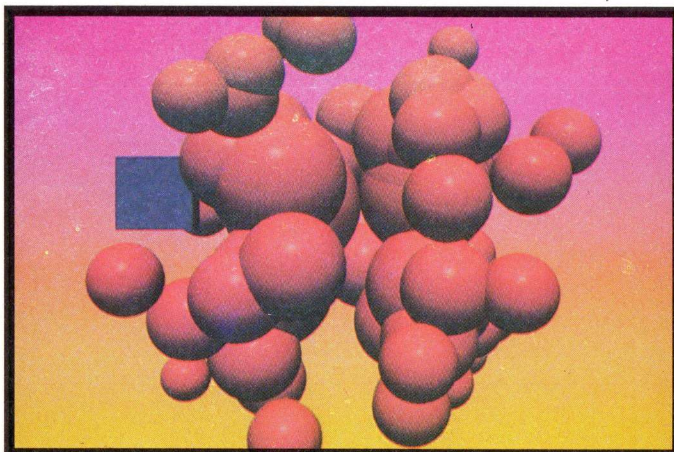
Immer mehr jedoch dienen Computer dazu, realistische Bilder zu erzeugen, dreidimensionale Welten zu erschaffen und zu simulieren, und hier liegt wohl auch die größte Faszination der Computergrafik.

Dieser Artikel will kurz und knapp wichtige Entwicklungen und Verfahren auf dem Sektor Computergrafik vorstellen. Literaturvorstellungen an anderer Stelle in diesem Heft sollen dem interessierten Leser die Möglichkeit geben, sich umfassender zu informieren.

Grundsätzlich gibt es zwei Techniken für die Erzeugung von Computerbildern: Vektor- und Rastergrafik. Rastergrafik kennen Sie alle von Ihrem Computer: Ein Rastergrafikgerät setzt Bilder aus einzelnen Punkten zusammen, die in einem Raster angeordnet sind. Um z.B. eine Linie in einem Raster zu zeichnen, muß der Computer feststellen, welche Punkte des Rasters zu der Linie gehören und welche nicht. Um einen Buchstaben zu zeichnen, muß dieser als Raster dargestellt werden. Je mehr Punkte der Bildschirm zur Verfügung stellt, je feiner also das Raster ist, desto besser kann die Bildqualität werden. Die Fehler, die dadurch auftreten, daß die Bildschirmauflösung nicht ausreicht, um die abzubildende Grafik exakt darzustellen, äußern sich in den typischen "Computertreppchen" bei schrägen Linien. Diese sogenannten Aliasing-Verzerrungen entsprechen übrigens genau denen, die auch bei der Digitalisierung von Musik auftreten.

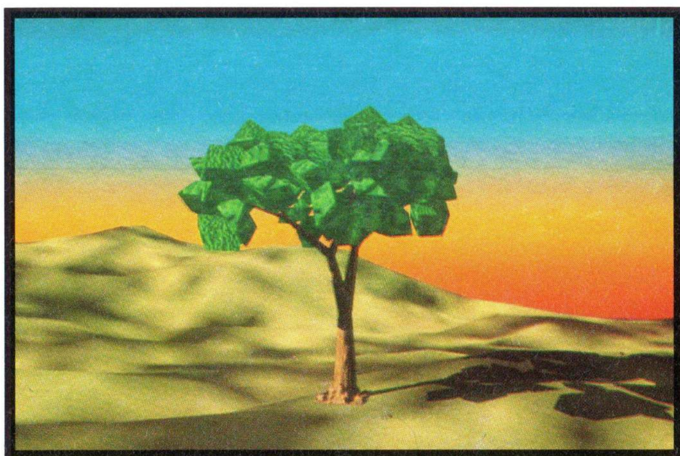
Die Darstellung von Formen, die im allgemeinen analog sind, also aus stetigen Elementen bestehen, in einem Raster ist nichts anderes als ein Digitalisierungsvorgang; das von Shannon im Jahre 1948 aufgestellte Abtasttheorem gilt für jede Art von analoger Information, auch für Bilder. Auf heutigen Rastergrafikgeräten ist die verzerrungsfreie Darstellung im allgemeinen noch nicht ohne weiteres möglich; die Auflösung ist einfach noch zu gering. Außer der Rasterauflösung ist auch die Anzahl der Farben, die sogenannte Farbauflösung, für die Bildqualität entscheidend. In der Natur gibt es im allgemeinen keine scharfen Farbkontraste, sondern nur weiche Farbübergänge. Um solche Übergänge auf dem Computer darzustellen und die Aliasing-Verzerrungen zu mindern, ist eine große Farbpalette notwendig. Professionelle Systeme arbeiten daher mit ca. 16 Millionen Farben.

Copyright '87, by Art Com Atelier für Computergrafik GmbH, Bremen.



Vektorgrafik

Vektorgrafiksysteme sind keine rein digitalen Systeme. Ihr Funktionsprinzip ähnelt eher einem Oszilloskop. Ein Vektorgrafikbildschirm setzt, ähnlich wie ein Plotter, ein Bild aus Linienzügen, die meistens gerade sind (aber bei analogen Systemen genausogut gebogen sein können), zusammen. Mit der steigenden Qualität der Rasterbildschirme haben Vektorsysteme immer mehr an Bedeutung verloren; es ist beispielsweise mit einem Vektordisplay nur sehr schwer möglich, einen Teil des Bildes nachträglich zu verändern, es muß immer das ganze Bild neu gezeichnet werden. Die maximale Geschwindigkeit ist durch die Steuergeschwindigkeit des Elektronenstrahls, der das Bild erzeugt, begrenzt. Gefüllte Flächen sind sehr schwierig zu zeichnen. Schließlich sind, der gefallenen Speicherpreise wegen, Rasterdisplays auch erheblich preisgünstiger als ihre Vektorkollegen. Ihrer extremen Auflösung wegen finden sie aber immer noch Anwendung im CAD-Bereich, wo häufig Drahtmodellzeichnungen genügen, gefüllte Flächen oder gar realistische Bilder also gar nicht gefragt sind.



Copyright '87, by Art Com Atelier für Computergrafik GmbH, Bremen.

Die Unterscheidung zwischen Raster- und Vektorgeräten findet man genauso im Bereich anderer Ausgabegeräte: Ein Matrixdrucker ist ein Rastergerät, ein Plotter (je nach Prinzip) eher ein Vektorsystem. Fast alle modernen Ausgabegeräte arbeiten heute, der größeren Vielseitigkeit und technischen Anspruchslosigkeit wegen, nach dem Rasterprinzip; die Palette reicht von der Satzmaschine bis zum Filmrecorder. Das sind Geräte, die die Aufzeichnung hochauflösender Bilder auf Filme erlauben, mit typischen Auflösungen von 2000*2000 bis 4000*4000 Punkten, bei 16 Millionen Farben. Mit solchen Anlagen, die bei guter Qualität einige 100.000 DM kosten, können z.B. Kinofilme direkt vom Computer erzeugt werden; der vor einigen Jahren wegen seiner Computeranimation bahnbrechende Walt Disney-Film "TRON" ist auf diese Weise erzeugt worden. Man zieht derartige Geräte im allgemeinen Videorecordern vor, weil die Qualität von Videoaufnahmen der eines 35mm-Films hoffnungslos unterlegen ist. Von diesen technischen Vorüberlegungen nun zur mehr inhaltlichen Seite. Wo liegen die prinzipiellen Unterschiede zwischen den verschiedenen Anwendungen der Computergrafik, und welche Techniken werden dafür benötigt ?

Malprogramme

Malprogramme sind die einfachste Form von Computergrafik, weil sie keinerlei "Einsicht" in den grafischen Sinn ihrer Aktionen haben müssen. Malprogramme sind eigentlich nur auf Rastergrafikbildschirmen sinnvoll. Ein solcher Bildschirm ist für ein Malprogramm eigentlich nur ein bestimmter Speicherbereich, der mit verschiedenen Operationen gefüllt beziehungsweise manipuliert werden kann. Wenn ein Malprogramm z.B. einen Buchstaben auf den Bildschirm zeichnet, ist dies im Grunde nur eine Kopierfunktion aus einem Speicherbereich, der eine Buchstabentabelle enthält, in den Bildschirmspeicher. Nach dieser Operation kann ein solcher Buchstabe nicht als Text verändert werden, sondern nur als eine Ansammlung von gesetzten oder gelöschten Punkten.

Allgemein kann ein Objekt, das einmal auf dem Bildschirm gelandet ist, nicht mehr als solches ediert werden, weil das Malprogramm den grafischen Sinn der gesetzten Punkte auf dem Bildschirm nicht kennt.

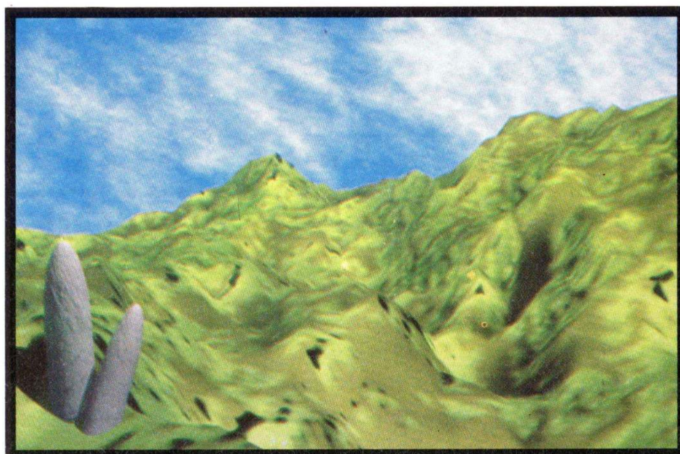
Ein Kreis ist nicht bekannt als die Information "An der Stelle (45,300) soll ein Kreis mit dem Radius 7 gezeichnet werden", sondern, wie gesagt, nur als eine Reihe von Punkten, deren Zusammenhang aber nicht bekannt ist. Daher kann der Kreis auch nicht ohne weiteres geändert werden; zuerst müssen alle beteiligten Punkte von Hand gelöscht werden.

Programme dieser Art haben heute eine ungeheure Funktionsvielfalt erreicht. Selbst auf Mikrocomputern wie dem Amiga, dem MacIntosh oder dem ST sind erstaunliche Bilder machbar. Im Profi-Bereich ist das bekannteste und verbreitetste System die Quantel Paintbox, die sogar über einfache Zeichentrickfunktionen verfügt. 16 Millionen Farben und Echtzeit-Farbdigitalisierung sind selbstverständlich. State-of-the-Art-Programm bei den Micros ist vielleicht Deluxe Paint auf dem Amiga. Dieses Programm müssen wir unseren Lesern an dieser Stelle wohl kaum noch vorstellen.

Objektorientierte Grafikprogramme

Malprogramme jeder Art laufen unter dem Oberbegriff "pixelorientierte Grafikprogramme", weil sie einzelne Pixel auf dem Bildschirm direkt beeinflußen. Im Gegensatz dazu stehen die sogenannten "objektorientierten Grafikprogramme". Ein solches Programm "weiß", was es tut; es speichert Objekte in einer verhältnismäßig ausgabeunabhängigen Form und wandelt diese Objekte erst dann in Pixelform um, wenn der Benutzer eine Änderung verlangt. Ein Kreis wird, um das obige Beispiel aufzugreifen, also tatsächlich als "Achtung, Kreis bei Punkt (45,300) mit Radius 7" gespeichert. Änderungen sind kein Problem: Der Computer kann problemlos das ganze Bild neu zeichnen, da er ja genau "weiß", welche grafischen Objekte sich auf dem Bildschirm befinden. Es muß dem Computer nur mitgeteilt werden, welches Objekt wie verändert

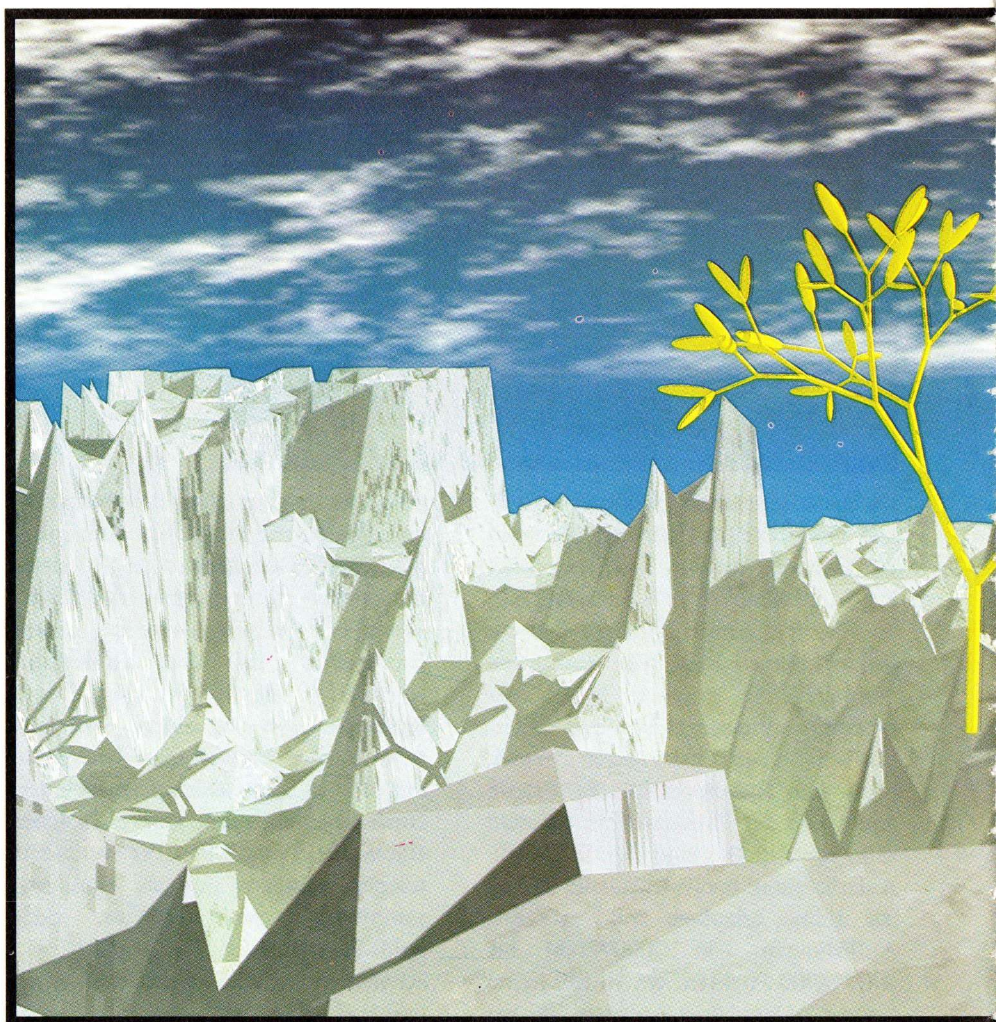
Copyright '87, by Art Com Atelier für Computergrafik GmbH, Bremen.



werden soll. Ein weiterer Vorteil ist, daß Speicherplatz eingespart wird. Ein Bild muß nicht mehr Bildschirmpunkt für Bildschirmpunkt auf Diskette gespeichert werden, sondern nur als Beschreibung der Objekte. Die Beschreibung "Kreis bei (45,300), Radius 7 mit Linienstärke 2" besteht aus 4 Zahlen. Die Pixel-Beschreibung des Kreises besteht aus den Koordinaten aller Pixel, die der Kreis berührt.

Nachteil ist allerdings, daß bei Änderungen fast immer das ganze Bild neu aufgebaut werden muß, da es bei überlappenden Objekten meist sehr schwierig ist, festzustellen, welche Teile welcher Objekte neu gezeichnet werden müssen. Da dauert ein kompletter Neuaufbau auch nicht länger...

CAD-Systeme werden aus verschiedenen Gründen eigentlich immer als objektorientierte Systeme konzipiert. Schließlich soll der Computer nicht nur die Zeichenarbeit erleichtern (also das Zeichenbrett verbessern), sondern auch die eigentliche Konstruktionsarbeit. Das Objekt "Getriebe" wird aus grafischen Grundobjekten, mit denen für technische Berechnungen auch physikalische Daten verknüpft sein können, zusammengesetzt, ein Zahnrad zum Beispiel aus kreisförmigen Scheiben, die mit Zähnen versehen sind. Der Computer malt also nicht nur "geistlos" Bilder, sondern kann Zeichnungen mit Hilfe von "Wissen" über das zu zeichnende Objekt erstellen. Am Beispiel Getriebe: In Verbund mit dem entsprechenden Rechenprogramm kann das Zeichenwerkzeug das gesamte Getriebe schnell neu zeichen, nachdem eine komplette Neudimensionierung wegen irgendwelcher geänderter Anforderungen notwendig wurde. Oder ein musischeres Beispiel: Gute Notendruckprogramme, die in der Lage sind, hochwertige musikalische Partituren zu drucken, speichern Musik nicht als Pixel-Grafik, sondern als eine Folge von Daten, welche die Musik repräsentieren. Daraus muß das Programm dann die Information entnehmen, wie das korrekte Notenbild auszusehen hat. Dies ist eine spezielle Form des allgemeinen Anwendungsgebietes "Desktop-Publishing". Um eine schnelle und komfortable Bearbeitung eines Layouts zu ermöglichen, wird eine Seite als Liste von Grafikobjekten gespeichert (auch Text ist ein Sonderfall von Grafik) und erst für die Ausgabe in das für das Ausgabegerät (z.B. eine Fotosatzmaschine) notwendige Format umgewandelt. Speziell für diese Art von Anwendungen wurden Seitenbeschreibungssprachen wie PostScript oder DDL entwickelt, die eine standardisierte, auflösungsunabhängige Übertragung und Speicherung von Layoutseiten erlauben. Auch für allgemeine Grafikanwendungen gibt es auflösungsunabhängige Normen wie z.B. das Grafische Kern-System, abgekürzt GKS, das für so ziemlich jeden ernstzunehmenden Rechner zur Verfügung steht und nicht nur die Übertragung von Zeichnungen, sondern im allgemeinen auch die Übertragung ganzer Applikationen erlaubt. Es ist eine Bibliothek von Grafikroutinen, die durch einen Gerätetreiber an nahezu jedes Ausgabegerät angepaßt werden kann. Dieser Treiber ist der einzige geräteabhängige Teil des Systems.



Copyright '87, by Art Com Atelier für Computergrafik GmbH, Bremen.

Universalität, Einfachheit und Effizienz

Damit wären wir an einem Punkt angelangt, der für professionelle Grafiksysteme von ungeheurer Wichtigkeit ist: Nahezu jedes System muß versuchen, die drei Forderungen Universalität, Einfachheit und Effizienz unter einen Hut zu bekommen. Leider schließen sich diese Forderungen meistens aus. Universelle Algorithmen sind im allgemeinen nicht besonders effizient, einfache Systeme selten universell. Universalität bedeutet zum einen möglichst weitgehende Geräteunabhängigkeit. Damit muß nicht unbedingt die Portabilität der Grafik-Applikation gemeint sein, sondern die Möglichkeit, verschiedene Ausgabegeräte ohne erhebliche Änderungen an der Applikation verwenden zu können. Eine Applikation ist sauber und universell programmiert, wenn sie in der Lage ist, ein Bild ohne zeitraubende



Umrechnung und ohne Qualitätsverlust z.B. auf einem Bildschirm und einer Fotosatzmaschine auszugeben.
Erreichen läßt sich dies, indem man alle Berechnungen in einem allgemeinen Koordinatensystem durchführt, den (meist) sogenannten Weltkoordinaten. Dies sind im allgemeinen Floating-Point-Zahlen, also der gesamte Rechenbereich des Computers. Erst der Gerätetreiber selbst übernimmt die Umrechnung in die Gerätekoordinaten des Ausgabegerätes. Nachteil: Nicht sehr effizient, da erstens mit unhandlichen Zahlen (Fließkomma-Arithmetik) und zweitens mit einer zusätzlichen Transformation gerechnet werden muß. GKS benutzt sogar drei verschiedene Koordinatensysteme: Die "Welt", die Zeichnung also, wird in einem (theoretisch) unbegrenzten Weltkoordinatensystem gespeichert. Vor der Ausgabe wird jede Grafik in sogenannte "Normalized Device Coordinates" (normalisierte Geräte-Koordinaten) umgerechnet, die auf der x- und y-Achse von 0 bis 1 reichen. Die Umwandlung in die aktuellen, endgültigen Gerätekoordinaten übernimmt dann eine letzte Transformation, wobei für professionelle Systeme Auflösungen bis zu 4000*4000 Punkten auf Bildschirmen in Gebrauch sind. Einfachheit bezieht sich vor allem auf die Bedienung, die Interaktion mit dem Benutzer. Ein Grafiksystem sollte einfach und intuitiv zu benutzen, dazu in seinen einzelnen Teilen konsistent (d.h., gleiche Benutzer-Aktionen sollten in verschiedenen Programmteilen vergleichbare Aktionen auslösen) sein. Die Effizienz trägt dazu durchaus bei, denn auch schnelle Antwortzeiten eines Rechnersystems tragen erheblich zum Komfort bei der Bedienung bei.

Die dritte Dimension

Dreidimensionale Computergrafik und vor allem der Versuch der Simulation der Wirklichkeit, beziehungsweise der Versuch, künstliche Wirklichkeiten zu schaffen, stellen besondere Anforderungen.
Als Ansatz kommt eigentlich nur ein objektorientiertes System in Frage; schließlich kann man 3D-Objekte nicht so ohne weiteres auf einem zweidimensionalen Bildschirm darstellen, dazu muß ein wenig gerechnet werden. Das Problem der Perspektive, oder allgemein der zweidimensionalen Darstellung der dreidimensionalen Welt (allgemein Projektion genannt) stammt natürlich nicht erst aus unserem Zeitalter. Bereits die Schöpfer der frühesten Höhlenmalereien waren mit diesem Problem konfrontiert, für das in der Kunstgeschichte später sehr verschiedene und sehr raffinierte Lösungen gefunden wurden. Besonders auffällig ist in dieser Hinsicht die Kunst des europäischen Mittelalters, die auf eine wirklich perspektivische Darstellung wieder weitgehend verzichtet. Spätere Werke haben den Kunsthistorikern Rätsel aufgegeben: Erst in einer neueren Studie, die an der TH Darmstadt mit Hilfe einer Computergrafik-Simulation (!) durchgeführt wurde, konnte das Problem der Perspektive in Raffaels 1511 entstandenem Fresko "Die Schule von Athen" halbwegs gelöst werden: Um den gewünschten Raumeindruck zu erhalten, hat der Maler wahrscheinlich einfach verschiedene Perspektiven für verschiedene Raumteile verwendet.
Das Problem jeder Perspektive, besonders für konstruktive Zwecke, ist die entstehende Verzerrung: Weiter entfernte Objekte erscheinen kleiner als näherliegende. Daher wurden Projektionen entwickelt, die bestimmte Objekteigenschaften in der 2D-Darstellung erhalten. Diese Darstellungen sind weniger realistisch, haben aber für CAD-Anwender oft den höheren Informationswert.

Objektbeschreibungen

Ein weiteres Problem für die Computergrafik ist die Beschreibung der Objekte. Die meisten Formen des "wirklichen Lebens" sind nicht gerade einfach zu beschreiben, jedenfalls nicht mit der notwendigen Exaktheit. Man kann daher extrem einfach beginnen: Das einfachste aller Objekte ist der Punkt. Die Verbindung zweier Punkte ist eine Linie. Mehrere Linien umschließen eine Fläche, mehrere Flächen ein Volumen. Man kann die

Copyright '87, by Art Com Atelier für Computergrafik GmbH, Bremen.



Oberfläche eines Objektes also aus kleinen Flächenstücken (Polygonen oder Vielecken) zusammensetzen. Leider erhält man damit nur brauchbare Darstellungen, wenn man die Polygone sehr klein macht, also sehr viele Polygone aneinandersetzt, was wiederum nicht sehr effizient ist. Andererseits lassen sich nur sehr einfache Körper wie Kugeln oder Zylinder auf einfache Weise mit leichtverständlichen Formeln exakt mathematisch beschreiben. Ein Verfahren, mit dessen Hilfe sich gebogene Flächen mathematisch exakt beschreiben lassen, wird also gebraucht. Das erste brauchbare System, das ein solches Verfahren verwendete, hieß "Unisurf" und wurde von dem französischen Mathematiker Bezier für die Autofirma Renault zum Entwurf von Karosserieteilen entwickelt. Die sogenannten Bezier-Kurven sind die mathematische Simulation eines alten mechanischen Werkzeuges: Früher benutzte man für die Formung von Kurven in Werkstätten elastische Metallstreifen, die an ihrem Anfang und Ende befestigt und über deren Länge verteilt einige Gewichte an Schnüren angebracht waren. Durch Verschieben der Gewichte auf dem Arbeitstisch konnte man den Streifen nahezu beliebig verformen. Bezierkurven verhalten sich ganz ähnlich: Die Form der Kurven wird durch sogenannte Kontrollpunkte bestimmt, die nicht auf der Kurve liegen, wohl aber ihren Verlauf bestimmen.

Eine andere, besonders im Maschinenbau beliebte Darstellungsform ist das CSG-Verfahren (Constructive Solid Geometry - konstruktive Volumen-Geometrie). Hierbei werden komplizierte Objekte aus einfachen Grundformen wie Kugel, Kegel oder Zylinder, die sich mathematisch problemlos beschreiben lassen, zusammengesetzt, wie im Holzbaukasten, nur mit dem Unterschied, daß man Objekte nicht nur an- und ineinander setzen, sondern sie auch voneinander abziehen kann, zum Beispiel um ein Loch in einem Rad zu erhalten. Dafür nimmt man zwei Zylinder, davon einen sehr flachen als Rad und einen, der den Lochdurchmesser besitzt. Diesen positioniert man nun an der Stelle des gewünschten Loches und zieht ihn vom Rad ab; als Ergebnis ist ein Loch im Zylinder. Im Grunde ist das nichts anderes als die Mengenlehre der Schulkinder - Schnitt- und Differenzmengenbildung usw. Obwohl die Operationen (sogenannte Bool'sche oder Mengenoperationen) recht einfach sind, kann der Rechenaufwand für die Darstellung ziemlich hoch sein, weshalb das Modell meistens nicht auf Micro-CAD-Systemen verwendet wird. Dafür ist der Speicheraufwand auch für komplexe Objekte extrem gering, weil statt der Berechnung des Endprodukts nur ein Baum, also die einzelnen primitiven Objekte und die gewünschten Verknüpfungen, gespeichert werden. Vor der Darstellung kommt man dann aber um das Berechnen nicht mehr herum.



Copyright '87, by Art Com Atelier für Computergrafik GmbH, Bremen.

Verdeckte Oberflächen

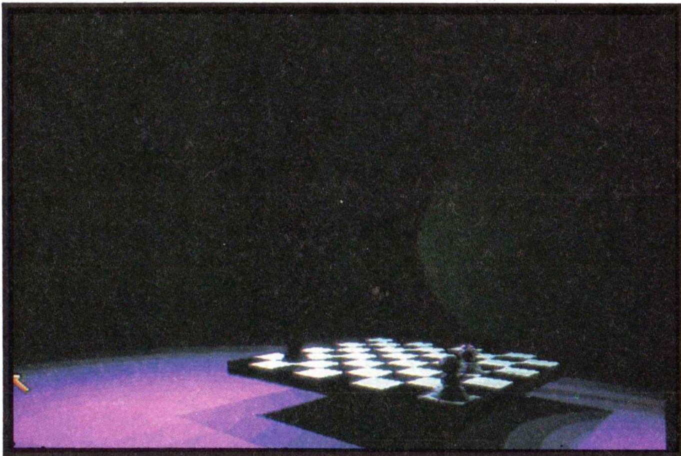
Das Problem der Darstellung: Man kann ja nicht einfach alle Linien, Punkte oder sonstigen Teile eines Objektes auf den Bildschirm projezieren, um ein ansehnliches Objekt zu erhalten. Vorher muß man erst einmal feststellen, welche Teile eines Objektes vom Blickpunkt des Betrachters aus überhaupt sichtbar sind und welche durch sich selbst oder andere Objekte verdeckt werden. Dieses Problem ist nicht ganz einfach und eigentlich nur für Polygon-Darstellungen effizient zu lösen. Es gibt eine ganze Reihe von Algorithmen, die alle ihr besonderes Anwendungsgebiet haben und außerdem eine Gemeinsamkeit: Entweder sie sind schnell, oder sie sind gut. Schlechte Algorithmen machen Fehler, das heißt, bei bestimmten Objekten sind Teile sichtbar, die eigentlich verdeckt sein müßten. Gute Algorithmen sind dafür langsam. Für jede Anwendung muß man einen Kompromiß suchen. Besonders verbreitet ist der sogenannte Painter-Algorithmus. Hierbei werden erst alle Polygone entsprechend ihrer Entfernung vom Beobachter sortiert und dann von hinten nach vorne gezeichnet. Dadurch werden die verdeckt (also hinten) liegenden Polygone von weiter vorne liegenden übermalt. Weil diese Technik auch beim Malen mit Ölfarben gebräuchlich ist, erhielt der Algorithmus seinen Namen.

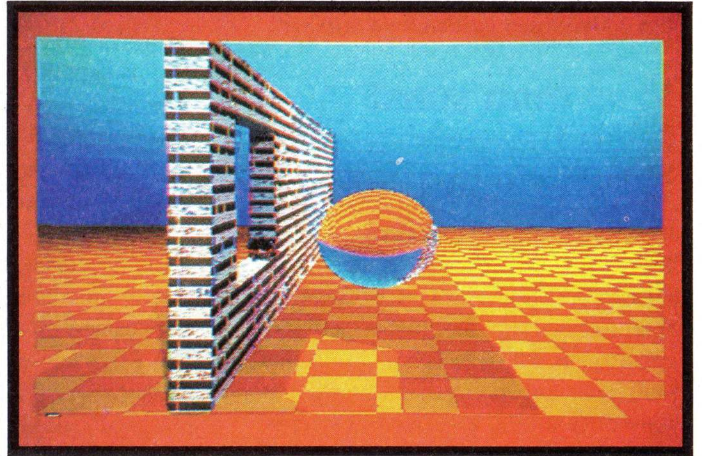
Die Qualität der Darstellung hängt nur von der Qualität der Sortierung ab - und auch hier gilt: Gute Sortierung braucht ihre Zeit.

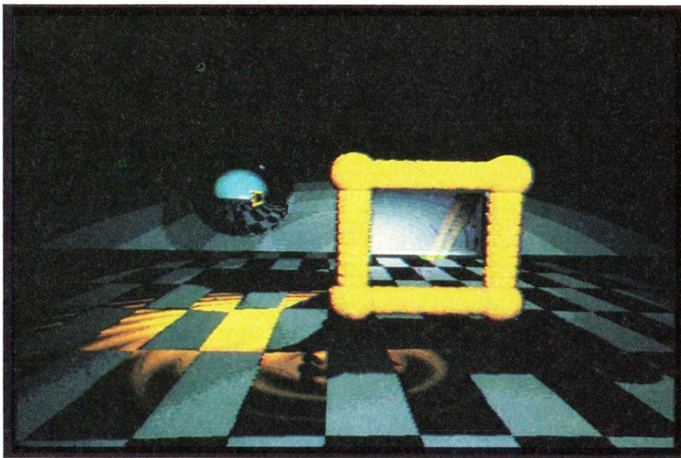
Schattierung

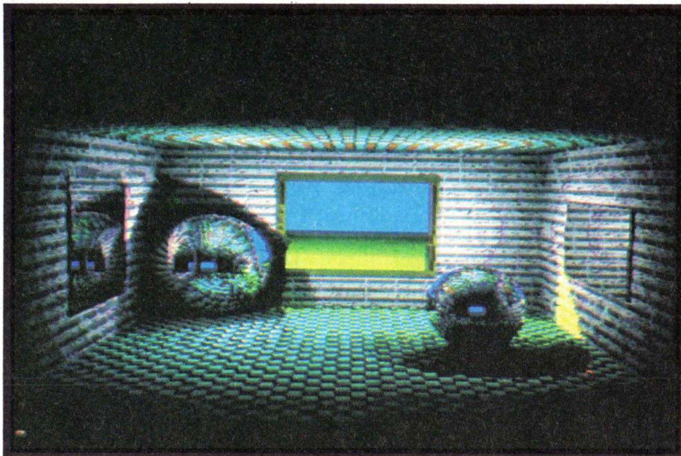
Um realistische Bilder zu erhalten, genügt es aber nicht, einfach nur alle sichtbaren Polygone in irgendeiner Farbe zu zeichnen. Natürliche Flächen zeigen immer von der Beleuchtung abhängige, kontinuierliche Farbschattierungen. Deshalb muß jetzt die Physik 'ran. Glücklicherweise liefert die Optik recht brauchbare Modelle über das Verhalten (sprich die Reflektion) von Lichtstrahlen an einer mehr oder weniger spiegelnden Oberfläche. Wenn dieses Verhalten bekannt ist, und auch der Winkel einer Fläche zu einer Lichtquelle, dann kann man die Schattierung dieser Fläche (also dieses Polygons) berechnen. Das bringt uns aber noch nicht viel weiter, denn jetzt ist immer noch jedes Polygon gleichmäßig gefärbt. Was tun? Im Laufe der Zeit wurden einige Verfahren entwickelt, die darauf basieren, daß man sich die Winkel der umliegenden Flächen zur Lichtquelle

GEGENÜBERSTELLUNG



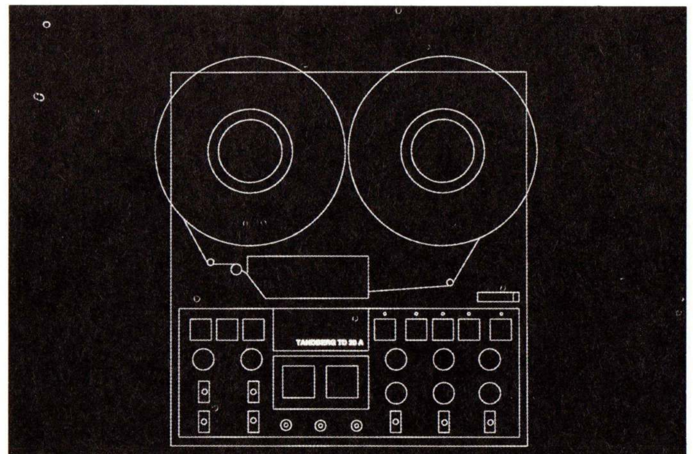






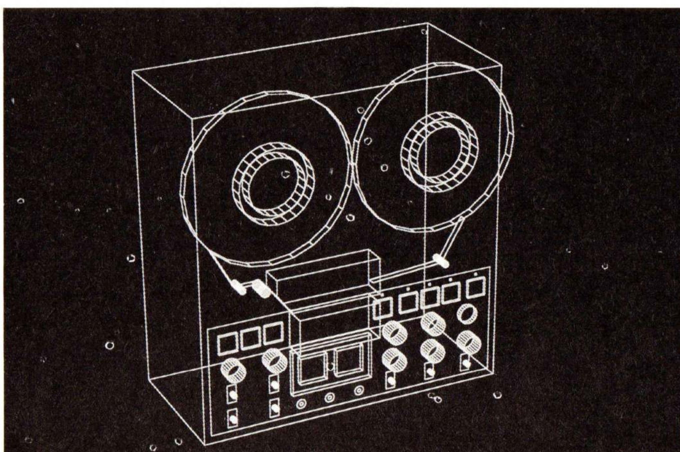
Auch mit einem Microcomputer wie dem Amiga ist es heutzutage möglich, fotorealistische Bilder zu berechnen und darzustellen.
Natürlich unterscheiden sich diese in der Qualität doch deutlich von den hier abgebildeten Werken der Computergrafik-Profis.
Einige solcher Amiga-Grafiken, die Sie auch auf den KICKSTART PD-Disketten finden, sollen dennoch - oder auch gerade deshalb - hier abgebildet werden.

vornimmt und von der Mitte des zu schattierenden Polygons aus diesen Winkeln immer mehr annähert. Daraus ergibt sich eine gleichmäßige Schattierungsänderung auch über Polygonkanten hinweg. Die bekanntesten Verfahren stammen von Gouraud und Phong, wobei besonders letzteres auch sehr gute Schlaglicht-Effekte auf spiegelnden Oberflächen erzeugt.
Wenn man Farbbilder erzeugen will, müssen die entsprechenden Berechnungen im allgemeinen für jede der drei Grundfarben Rot, Grün und Blau einzeln durchgeführt werden, um den endgültigen Farbwert zu erhalten.


Copyright '87, by: Workshop-Artline, Frankfurt.

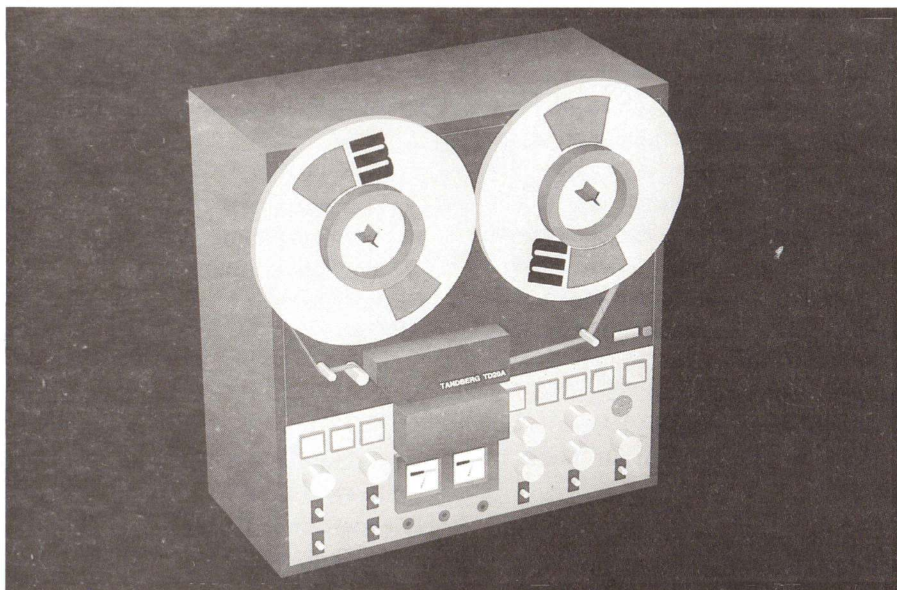
Was man sonst noch braucht... Ray Tracing

Jetzt fehlen aber immer noch Schatten und Transparenzeffekte. Es ist extrem schwierig, diese Art von Effekten in eine Darstellung der bisherigen Art einzubauen. Für wirklichen Realismus muß man auf ein anderes Verfahren ausweichen, das einerseits wie eine Brute-Force-Methode erscheint, andererseits aber eine Art eierlegende Wollmilchsau ist, weil sowohl das Ausblenden von verdeckten Oberflächen, als auch Spiegelungen, Schattierungen, Transparenz und Schatten in einem Aufwasch erledigt werden. Nachteil: Keinerlei Eigenschaften der Objekte selbst werden ausgenutzt, die Rechenzeit ist extrem. Das Verfahren nennt sich Ray Tracing, zu Deutsch Strahlverfolgung.
Wie dieses Verfahren funktioniert und welche Vorteile es bietet, wird an anderer Stelle in diesem Heft ausführlich erläutert und deshalb hier ignoriert. Es sei nur erwähnt, daß Ray Tracing von der Bildqualität (Realismus der Darstellung usw.) her die mit Abstand besten Ergebnisse bringt. Auch die Implementierung ist im Vergleich zu anderen Verfahren extrem einfach.
In neuester Zeit wurden auch Algorithmen entwickelt, die das größte Manko des Verfahrens relativieren, nämlich den extremen, exponential mit der Objektanzahl ansteigenden Rechenaufwand. Inzwischen gibt es Ray Tracing-Routinen, die annähernd konstante Rechenzeit, fast unabhängig vom Bild, benötigen. Nahezu alles, was an interessanten Computeranimationen veröffentlicht wird, ist mit Hilfe von Ray Tracing-Verfahren erzeugt worden. Das bisher vielleicht perfekteste Beispiel ist der Film "Luxor jr.", der auf der Linzer Ars Electronica '87 den großen Preis für Computeranimation gewonnen hat und als erster 3D-Computertrickfilm für einen Oscar vorgeschlagen wurde. In dem Film spielen eine große und eine kleine Schreibtischlampe miteinander Ball, wobei sich die Lampen sehr "menschlich" bewegen. Schließlich hüpft die kleine Lampe auf den Ball (die Kinder treiben es ja immer zu toll...), der natürlich auch prompt einen Platten bekommt. Aber schon findet sich ein Ersatzball, und weiter geht's, worüber die große Lampe nur noch den Kopf schütteln kann; Ja,ja, diese Kinder...
Der Detailreichtum und vor allem die Bewegungsqualität dieses Films sind faszinierend.


Copyright '87, by: Workshop-Artline, Frankfurt.

Werbung und Animation - 3D-Computergrafik im Einsatz

Das Hauptanwendungsgebiet für 3D-Grafiksysteme liegt traditionell im Film- und Fernsehbereich. Doch auch die Wissenschaft macht sich die Möglichkeit der realistischen Simulation unsichtbarer Vorgänge zu nutze, so zum Beispiel die Molekularbiologie, die mit Computern chemische Vorgänge simulieren und jetzt endlich auch darstellen kann. Besonders berühmt in dieser Hinsicht ist der Film "Die Entstehung des Lebens", in dem der Programmierer Nelson Max die Entstehung eines DNS-Moleküls auf die Leinwand zaubert. Auch der Flug durch die Ölkanäle eines laufenden Motors zeigt die Stärken der Computergrafik - Unsichtbares sichtbar zu machen.
In der Medizin kann ein Computer verwendet werden, um aus den zweidimensionalen Aufnahmen eines Tomographen ein dreidimensionales Bild des untersuchten Körperteils zu machen. Dabei ist es möglich, Bilder



Copyright ´87, by:Workshop-Artline, Frankfurt.

zu erzeugen, die aussehen, als seien sie frisch seziert. Jeder Muskel tritt deutlich wie im Anatomielehrbuch hervor. Auf diese Weise hat es der Arzt leichter, Krankheiten zu finden.

Aus Konstruktion und Design ist der Grafikcomputer nicht mehr wegzudenken. Er erlaubt es z.B. dem Architekten, einen Gang durch ein Haus zu machen, für das noch nicht einmal ein Grundstück existiert. Dem Bauherrn können so Fehlplanungen leichter erspart bleiben.

Vom technischen Design zur Gebrauchsgrafik. In der Werbung zeigt sich ein weiteres großes Anwendungsgebiet. Vom zweidimensionalen Layouten mit Desktop-Publishing-Systemen abgesehen, werden 3D-Systeme sehr gerne verwendet, um die besonderen Effekte zu produzieren, die sonst mühsam mit der Spritzpistole oder anderen Techniken erstellt werden müssen. Hauptvorteil ist hierbei die Tatsache, daß der Grafiker Änderungswünsche des Kunden erheblich schneller und müheloser realisieren kann, als mit konventionellen Techniken. Dafür liegen die Kosten auch noch erheblich höher. Das am weitesten verbreitete Grafiksystem der unteren Preisklasse, das in der Lage ist, professionelle Grafiken zu erzeugen, ist der Cubicomp PictureMaker. Er basiert auf einem IBM-AT, der durch eine ca. 100.000 DM teure Hardwarebox erweitert wird. Um auf vernünftige Geschwindigkeit zu kommen, kann man das System für ungefähr 50.000 DM mit einer Beschleunigungsbox versehen. Die Gesamtkosten liegen jedoch noch weit höher: Ein Filmrecorder (ab 25.000 DM, sinnvoll ab ca. 80.000 DM) wird gebraucht, ein Scanner zum Einlesen von Grafiken und ein Videodigitalisierer für Kamerabilder. Will



Copyright ´87, by: Workshop-Artline, Frankfurt.

man auch einfache Animationen rechnen, wozu das System durchaus in der Lage ist, wird ein einzelbildaufnahmefähiger Videorecorder gebraucht (U-Matic High Band, ab ca. 40.000 DM). Sie sehen, selbst die einfachste Profiausstattung ist nicht ganz billig. Videoeffekte kann man mit diesem System übrigens nicht berechnen. Die mittlere Preisklasse beginnt bei Animationssystemen bei ca. 500.000 DM. Damit kann man dann aber auch Ray Tracing-Filme von hervorragender Qualität rechnen. Herausragende Anbieter sind Wavefront und Silicon Graphics. Nur eine einzige europäische Firma ist auf dem Markt nennenswert vertreten, die französische Firma TDI. Natürlich können alle diese Systeme keine Ray Tracing-Bilder in Echtzeit produzieren. Dazu wären, selbst bei Anwendung modernster Algorithmen, mindestens 600-1000 MFlop (Millionen Fließkomma-Operationen pro Sekunde) Rechenleistung erforderlich. Die gängige VAX unter Unix kommt mit viel Glück auf 1 Mflop.

Objekte werden im allgemeinen zuerst als Drahtmodelle auf dem Bildschirm erzeugt. Bei einem Drahtmodell sieht man nur die Umrisse oder Konturen des Objektes. Diese kann man dann in Echtzeit auf dem Bildschirm bewegen. Auch die gewünschten Lichtquellen können plaziert und ausgerichtet werden. Für die eigentlichen "Aufnahme" steht dann eine "Kamera" zur Verfügung, die durch die Szenerie gefahren werden kann. Der Realismus bei der Kamerasimulation geht bei manchen Systemen so weit, daß sogar die Tiefenschärfe des Objektives berücksichtigt wird.

Es wurden in der Vergangenheit auch zahlreiche "Regisseursprachen" entwickelt, mit deren Hilfe der Regisseur dem System seine Anweisungen geben kann. Diese Sprachen sind aber im allgemeinen nicht sehr interaktiv.

Wenn die Drahtmodellfassung einer

Szene fertig ist, kann der Rechner Bild für Bild nach dem Ray Tracing-Verfahren berechnen. Das kann unter Umständen Stunden dauern, je nach Komplexität und, vor allem, je nach dem verwendeten Rechner. Glücklicherweise müssen Sie nicht die ganze Zeit danebensitzen und einen Kaffee nach dem anderen trinken, der Computer ist durchaus in der Lage, selbstständig mit Filmrecorder oder Videorecorder zu kommunizieren und sie zur Mitarbeit zu motivieren. Die Produktion selbst der einfachsten Animationen kann leicht eine oder mehrere Wochen dauern. Preise von 4.000-10.000 Dollar pro Sekunde (eine Sekunde Animation sind je nach Verwendung und Fernsehnorm nur 24, 25 oder 30 Einzelbilder!) Animation sind durchaus üblich.

Das bisher leistungsfähigste Bildrechensystem ist "Pixar", das von der amerikanischen Lucas Film gebaut wird. Auch "Luxor jr." oder die Glasritter-Sequenz aus Steven Spielbergs "Das Geheimnis des verlorenen Tempels" wurden mit Pixar berechnet. Weniger aufwendig ist dagegen die 3D-Bearbeitung von Videobildern, die wohl jeder aus Videoclips kennt: Fernsehbilder, die auf Trommeln projeziert sind, die umgeblättert werden wie Buchseiten oder durch den Raum fliegen wie Teppiche (fliegende natürlich). Dennoch, wenn solche Effekte in Echtzeit erzeugt werden sollen, wird es teuer. Das Quantel Mirage System, das meist für diese Aufgaben verwendet wird, kostet weit über 1 Million DM. Daher ist es etwas billiger, diese Effekte in Fast-Echtzeit im Studio zu erzeugen.

Kritisches

Die ganze Animationsbranche und zum Teil auch die Werbegrafik, soweit sie Computergrafik verwendet, ist im Moment noch durch starke Tendenzen zur billigen Effekthascherei gekennzeichnet. Das liegt zum größten Teil daran, daß an den Geräten häufig eher Techniker als Künstler, mehr Pro-



Copyright '87, by: Workshop-Artline, Frankfurt.

Zu den Grafiken:
Die in diesem und dem Ray-Tracing-Artikel in diesem Heft abgedruckten Grafiken wurden uns von den Firmen:

ArtCom -Atelier
für Computergrafik
Bremer Innovations- und
Technologiezentrum
Fahrenheitstraße 1
2800 Bremen 33

und:

Workshop-Artline
Düllberg & Brendel
Friedrichstraße 10-12
6000 Frankfurt 1

freundlicherweise zur Verfügung gestellt. Hierfür möchten wir an dieser Stelle noch einmal danken.

Die Firma ArtCom befaßt sich speziell mit der Erzeugung von Ray Tracing-Grafiken auf einem mit einem speziellen Grafiksystem erweiterten IBM AT und selbstgeschriebener Software. Die Firma Artline arbeitet ebenso mit selbstverfaßter Software auf einem erweiterten IBM AT, wobei die abgedruckten Bilder einer PaintBox entstammen. Neben der Herstellung von Computergrafiken unter anderem für Werbung und Unternehmenspräsentation vertreiben beide Firmen das von ihnen verwendete Designpaket (Hardware und Software), wobei sich die Systeme in beiden Fällen in einer Preisklasse von (je nach Ausführung) ca. 120000.- bis 150000.- DM bewegen.

VON ANDREAS DIEBOLD

EINFACHE DARSTELLUNGS-WEISEN

Der Amiga kennt mannigfaltige Möglichkeiten, mit seinem Benutzter in Kontakt zu treten. Die hohe Geschwindigkeit ermöglicht das Arbeiten mit mehreren Fenstern, die schnelle Verwaltung des Eingabegerätes Maus, natürlich auch der Tastatur, und vielem mehr.
Die Verwaltung all dieser Kleinigkeiten, in einer Systemumgebung, welche obendrein noch multitaskingfähig ist, stellt natürlich schon eine gewisse Komplexität dar.
Wer hier alle Ein- und Ausgabemöglichkeiten in totaler Eigenregie verwalten will, der hat ein ordentliches Stück Arbeit vor sich. Daher haben sich die Systementwickler etwas einfallen lassen, was sie Intuition nannten, und in das gewaltige Bauwerk der Amiga-Systemsoftware einfügten.

Intuition hilft uns Programmierern beim Verwalten von Bildschirmspeichern, welche der User in Form von Screens und Windows vor sich hat. Außerdem kümmert es sich noch ein wenig um die Eingabegeräte, in Verbindung mit gewissen Tasks und Windows. Doch sollte man sich über diese Hilfe nicht zu früh freuen, da diese gigantische Verwaltungsarbeit durch Intuition zwar einfacher, aber für den Amiga-Einsteiger nicht unbedingt einfach wird.

Dafür aber wird man mit einer recht umfangreichen Kontrolle belohnt, die man allen möglichen Anforderungen anpassen kann.
Im folgenden wollen wir uns den Umgang mit den wesentlichen Dingen anschauen, welche uns durch Intuition angeboten werden.
Entscheidend ist in der Kommunikation zwischen Mensch und Computer, beim momentanen Stand der Technik, die visuelle Ausgabe der Maschine. Hier erkennt der User den momentanen Status des Geräts und kann diesem, mittels Tastatur und Maus, Befehle geben. Daher baut auch Intuition auf einer optischen Grundlage auf. Diese ist das Window. Ein Window steht mit einem Task, also einem ablaufenden Programm in Verbindung, gibt dessen Informationen aus, und nimmt in Zusammenhang mit den Eingabegeräten Informationen für das Programm an. Da es möglich ist, mehrere Windows zu erzeugen, und sich von diesen verschiedenartigste Aufgaben erledigen lassen kann, muß man schon genau spezifizieren, was für ein Window gewünscht ist. Zusätzlich zu den Windows gibt es noch Screens. Der grundsätzliche Unterschied: Ein Screen ist der eigentliche Bildschirmspeicher, bzw. dessen Darstellung. Dem sei gleich angemerkt, daß ein Bildschirmspeicher im Amiga in einer BitMap organisiert ist. Das heißt, für jedes Pixel steht erst einmal ein Bit im Speicher, und zwar in einem zusammenhängenden Speicherblock, dessen erstes Bit das Pixel ganz links oben repräsentiert. Weiter geht es in der Waagerechten, von links nach rechts, Reihe für Reihe von oben nach unten. Das letzte Bit im Speicherbereich ist also das Pixel unten rechts. Da ein Bit bekannterweise nur zwei Zustände kennt, wird ein solcher Speicherbereich bei der Darstellung von mehr als zwei Farben noch mehrmals benutzt, beim Amiga im Normalfall maximal fünfmal. Benutzt man also beispielsweise fünf Speicherbereiche, so steht jedes erste Bit eines jeden Speicherbereichs für das linke obere Pixel, usw.. Nun wird ein Pixel also nicht mehr nur durch zwei mögliche Zustände repräsentiert, son-

dern durch 2 hoch 5, also 32. Dies wiederum heißt, daß 32 Farben ansprechbar sind.
Dieser Wert für jedes Pixel, quer durch die Speicherbereiche - nennen wir sie nun wieder BitPlanes - ist die Nummer eines Farbregisters, das jene Farbe, aus 4096 möglichen, repräsentiert, in der das Pixel dargestellt werden soll.
Ein Window ist immer ein Teil eines Screens. Ein Window benutzt normalerweise keinen eigenen Bildschirmspeicher, sondern den des Screens. Es ist also sozusagen ein Ein- und Ausgabegerät auf einem Bildschirmspeicher, sprich: Screen. Auch ein Screen verlangt eine genaue Definition, da es auch von seiner Sorte mehrere geben kann. Er ist zudem für die Anzahl und Art der Farben und die Pixelauflösung verantwortlich, nach denen sich alle Windows innerhalb eines Screen richten müssen. Dank der Hardware ist es möglich, auf dem Monitor mehrere Screens mit unterschiedlichen Farbpaletten und Auflösungen zur gleichen Zeit zu sehen.
Doch zurück zum eigentlichen Thema. Wie kommt der Programmierer zu Screens und Windows?
Ein Screen, der sogenannte WorkBenchScreen, wird von Intuition beim Starten von CLI oder Workbench immer zur Verfügung gestellt. Er hat auf einem Pal-Amiga von Haus aus eine Auflösung von 640 * 256 Bildschirmpunkten und stellt vier Farben dar. Auf ihm findet man auch das CLI-Window für Ein- und Ausgabe von DOS-Befehlen. Dieses Window hat eine Sonderfunktion, da es das Standard-I/O-Window darstellt. Daher kann man dieses Window zur einfachen Ein- und Ausgabe von Text benutzen, so wie man es von anderen Rechnern gewohnt ist, in C also z.B. mit den Funktionen printf(),scanf(), etc.. Um einem Programm aber sein eigenes Medium zu geben, welches auch die besonderen Eigenschaften des Amiga nutzt, brauchen wir ein Window. Dieses darf sich, wie schon angedeutet, auf einem eigenen neuen Screen oder aber dem WorkBenchScreen befinden.
Will man ein Window erzeugen, muß man dem Programm vorab Zugriff auf Teile des Betriebsystems geben. Da dieses in Libraries organisiert ist, geschieht dies einfach mit der Funktion OpenLibrary(). Die nötigen Include-Dateien erhält man durch den Präprozessoraufruf"#include<intuition/intuitionbase.h>". Die zu öffnenden Libraries sind die <intuition.library> und die <graphics.library>. Die erste sollte ihre Bedeutung im Namen tragen, und die zweite, die sich, oh Wunder, auf die Grafik bezieht, wird benötigt, da die Erzeugung von Bildschirmdarstellungen immer über den Weg der Grafik geht.
Wer das bisher Genannte getan hat, dem steht nun eine Struktur <NewWindow> zur Verfügung. Diese gilt es nunmehr zu initialisieren. Es folgt eine Beschreibung der Elemente dieser Struktur, mit einer Erklärung der möglichen Werte.

SHORT LeftEdge,TopEdge;

Hier werden die Koordinaten der linken oberen Ecke des Windows angegeben. Man bezieht sich dabei auf den Screen, in dem das Window gezeigt wird.

SHORT Width,Height;

Die Breite und Höhe des Windows, von der oberen linken Ecke aus.

UBYTE DetailPen,BlockPen;

In der Farbe <DetailPen> werden WindowGadgets und der Windowname dargestellt. <BlockPen> ist die Farbe des Rahmens. In beiden Fällen sollte eine gültige Farbnummer übergeben werden, wobei <NULL> immer die Hintergrundfarbe ist. Man kann auch den Wert -1 zuweisen. In diesem Fall werden die Farben für <DetailPen> und <BlockPen> vom jeweiligen Screen übernommen.

ULONG IDCMPFlags;

Dieses Langwort ist eine Bitmaske, d.h. jedes Bit steht für eine bestimmte Eigenschaft des I(ntuition) D(irect) C(ommunication) M(essage) P(ort) , der einen Verbindungskanal, der zwischen Intuition, bzw. dem Window, und dem Programm besteht, darstellt. Die Headerfiles bieten Wortdefinitionen für die einzelnen Bits an, was die Sache sehr veranschaulicht. Da diese Sache sehr umfangreich ist, ist eine genauere Erklärung im Abschnitt IDCMP zu finden. Will man keine IDCMP's nutzen, so ist hier einfach <NULL> einzusetzen.

ULONG Flags;

Dieses Element ist genauso aufgebaut wie das vorhergehende, aber es ist für äußerliche Merkmale zuständig, deren Bedeutungen wie folgt lauten:

SIMPLE_REFRESH

Ein Window dieser Art verliert seinen Inhalt, sobald es von einem anderen überlagert oder in der Größe verändert wird. Auch bei einer Verschiebung kann nur selten etwas an die neue Position gerettet werden.

SMART_REFRESH

Ein Window dieser Art behält seinen Inhalt bei jeglicher Überlagerung durch andere Windows. Auch ein Vergrößern übersteht es schadlos. Beim Verkleinern geht das verloren, was danach nicht mehr zu sehen ist. Natürlich benötigt dieses Verfahren mehr Speicherplatz als <SIMPLE_REFRESH>, da Bildbereiche zwischengespeichert werden müssen.

SUPER_BITMAP

Wie der Name sagt, bekommt das Window eine eigene BitMap, also einen eigenen Bildschirmspeicher. Dadurch ist der Inhalt vollkommen sicher, aber der Speicherverbrauch am höchsten. Soll dies benutzt werden, so ist in der NewWindow-Struktur auch ein Zeiger auf eine sinnvolle BitMap-Struktur zu setzen. Das Listing zu diesem Artikel beinhaltet ein Superbitmap-Window.

BACKDROP

Ein Backdrop-Window wird immer als Hinterstes geöffnet, und ist auch nicht in den Vordergrund zu bringen. Es ist nur das WINDOWCLOSE-Gadget möglich, das Window ist also nicht verschiebbar oder in der Größe veränderbar. Kein normales Window kann hinter dieses gelangen.

WINDOWSIZING

Mit diesem Flag ist das Window in seiner Größe veränderbar, so wie man das auch vom CLI-Window her kennt.

WINDOWDEPTH

Ermöglicht das in Vorder- und Hintergrundbringen des Windows.

WINDOWCLOSE

Setzt das Windowclose-Gadget in die Titlebar des Windows. Die Kontrolle übernimmt man mit Hilfe der IDCMP.

WINDOWDRAG

Das Window kann mit der Maus verschoben werden.

GIMMEZEROZERO

Wird in ein Window gezeichnet, so ist normalerweise der Koordinatenpunkt

0,0 die oberste linke Ecke des Windows. Dabei geht man natürlich das Risiko ein, die Titlebar oder den Rahmen zu übermalen. Mit diesem Flag wird der Koordinatenbezug innerhalb der Titlebar und des Rahmens gesetzt, doch verlangsamt diese Methode den Bildaufbau und kostet mehr Speicher.

BORDERLESS

Ein Window mit diesem Flag hat keinen Rahmen.

ACTIVATE

Alle neu geöffneten Windows, außer den BACKDROP-Windows, liegen zuerst im Vordergrund; mit <ACTIVATE> werden sie auch gleich aktiv.

NOCAREREFRESH

Normalerweise schickt Intuition immer eine Nachricht über IDCMP, wenn das Window Bestandteile durch Verschieben o.ä. verloren hat. <NOCAREREFRESH> unterdrückt dies.

REPORTMOUSE

Soll die Position der Maus über IDCMP übertragen werden, muß dieses Flag gesetzt sein.

RMPTRAP

Ist ein Window mit diesem Flag aktiv, so öffnet der rechte Mausknopf die Menübar nicht mehr.

struct Gadget *FirstGadget;

Soll das zu erzeugende Window keine eigenen Gadgets besitzten, so wird dieses Element <NULL>. Eigene Gadgets zu kreieren würde den Umfang dieses Artikels sprengen, der sich auf die Anwendung der grafischen Möglichkeiten reduzieren soll.

struct Image *CheckMark;

Dieser Zeiger verweist auf eine Struktur, die das Aussehen eines Symbols definiert, welches in Menüs schaltbare Zustände anzeigt. Gemeint ist damit z.B. der Haken, der im Programm NotePad im Menü jenen Font mit einem Haken bestätigt, der momentan aktiv ist.

Das System bietet als Voreinstellung diesen Haken. Will man ihn nicht benutzen, so ist hier <NULL> zu setzen.

UBYTE *Title;

Der Name des Windows wird als <unsigned char pointer>, der auf einen mit einem Nullbyte abgeschlossenen String zeigt, angegeben. Der in dem String enthaltene Name wird dann bei der Darstellung des Windows angezeigt. Benutzter von Aztec-C müssen dafür Sorge tragen, daß wirklich ein <unsigned char pointer> übergeben wird. Dies kann mit Hilfe des Cast-Operators (STRPTR) geschehen.

struct Screen *Screen;

Der Zeiger auf den Screen, in welchem das Window dargestellt werden soll. Den Zeiger erhält man nach dem Öffnen eines Screens mit OpenScreen(). Soll das Window auf dem WorkBenchScreen erscheinen, so ist hier einfach <NULL> zu setzen.

struct BitMap *BitMap;

Ein Window kann einen eigenen Bildschirmspeicher haben, der auch als Bitmap organisiert ist. Will man diese Möglichkeit nutzen, so ist in Zusammenhang mit dem Flag <SUPER_BITMAP> hier ein Zeiger auf die windoweigene BitMap anzugeben.

Im Normalfall steht auch hier einfach <NULL>.

SHORT MinWidth, MinHeight, MaxWidth, Maxheight;

Dies sind die sogenannten Windowlimits. Wird ein Window in seiner Größe verändert, so ist dies nur im Rahmen der hier gegebenen Werte möglich. Setzt man diese Variablen auf Null, so werden die oben genannten Eröffnungswerte eingesetzt.

USHORT Type;

Dieses Element bezieht sich nochmals auf den Screen, auf dem das Window dargestellt wird. Wird der WorkBench-Screen benutzt, so muß hier<WBENCHSCREEN> stehen, bei einem eigenen Screen <CUSTOMSCREEN>.

Nachdem diese Struktur entsprechend den eigenen Wünschen initialisiert wurde, wird ihre Anfangsadresse an die Funktion OpenWindow() übergeben. Ist alles korrekt, erhält man einen Zeiger auf die zu dem neuen Window gehörende Window-Struktur zurück. Diese Struktur wird im weiteren Verlauf des Arbeitens mit Windows noch von Bedeutung sein, da sie Elemente enthält, die unter anderem für grafische Ausgabe notwendig sind. Im Falle eines Fehlers, der durch falsche Parameter in der NewWindow-Struktur oder durch mangelnden Speicherplatz entstehen könnte, liefert OpenWindow() <NULL> zurück.

Genauso arbeitet die Funktion OpenScreen(), welche zum Öffnen eines Screens dient. Sie bekommt die Adresse einer Struktur NewScreen übergeben, welche folgende Elemente enthält:

SHORT LeftEdge,TopEdge, Width,Height;

Hier werden wiederum die Ausmaße festgelegt, die der Screen haben soll. Wichtig zu wissen ist dabei die Tatsache, daß Screens mitunter mehr auf dem Bildschirm überdecken, als aus den hier gegebenen Größen ersichtlich wird. Diese Tatsache beruht auf der Darstellungsweise von Bildschirmspeichern, mit der die Hardware arbeitet. So überdeckt ein Screen, egal, wie er in der Größe dimensioniert ist, in voller Breite den Bildschirm, von seiner obersten waagerechten Reihe aus nach unten. Er überdeckt dabei natürlich alles, was dahinter liegt, also auch andere Screens. Bei der Überdeckung dieser muß man sich die Ausbreitung nach unten unbegrenzt vorstellen. Es kann also niemals ein Teil eines im Hintergrund liegenden Screens unterhalb der obersten Reihe eines im Vordergrund liegenden Screens sichtbar sein.

SHORT Depth;

Die Anzahl der BitPlanes des Screens. Aus der Anzahl der BitPlanes ergibt sich die Anzahl der auf dem Screen zur Verfügung stehenden Farben, nach der einfachen Rechnung "Anzahl der Farben = 2 hoch Anzahl der BitPlanes". Maximal 5 BitPlanes, also 32 Farben, sind möglich. In zwei besonderen Darstellungsmodi, nämlich <DUALPF> und <HAM> sind auch 6 BitPlanes möglich (Siehe ViewModes).

UBYTE DetailPen,BlockPen;

DetailPen: Die Farbnummer, in der Gadgets und der Text der Titlebar dargestellt werden. BlockPen: Die Farbnummer der Titlebar selbst.

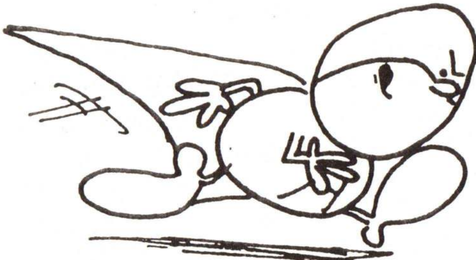
SHORT ViewModes;

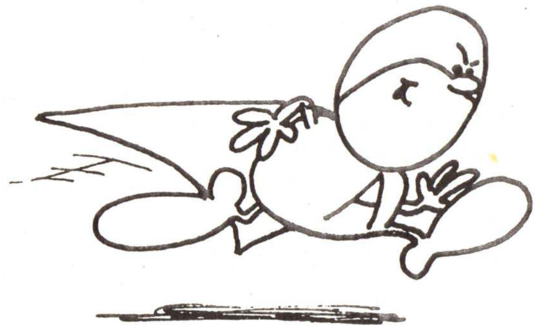
Die hierin enthaltenen Flags veranlassen die Hardware, die Darstellung auf eine bestimmte Weise erfolgen zu lassen. Es können ein oder mehrere Flags gesetzt werden:

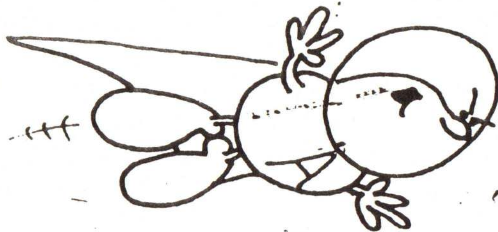
AMIGA MAN

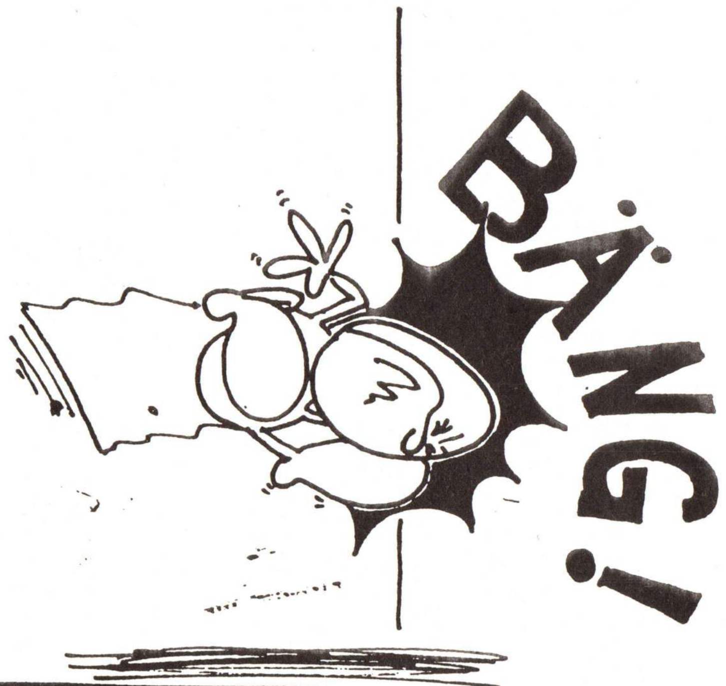
ABENTEUER IM AMIGA!

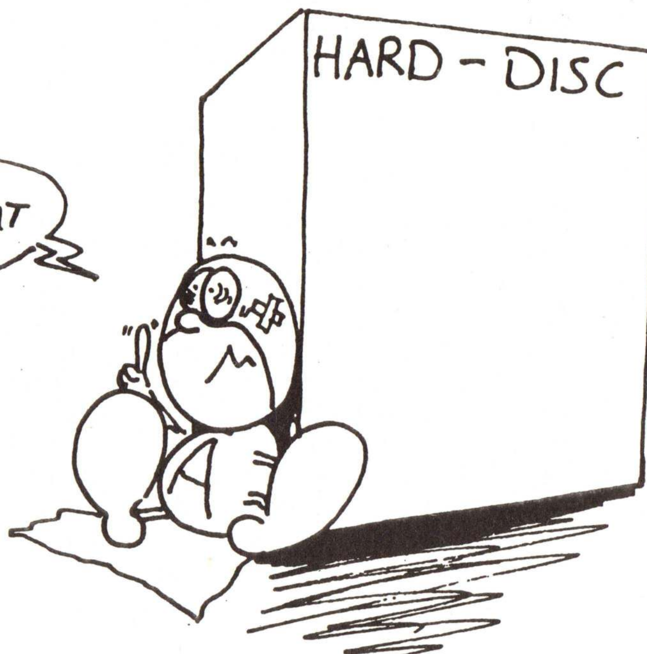












Wichtiger Hinweis

Folgende Video- und Computerspiele sind von der Bundesprüfstelle, Bonn, indiziert:

Battlezone
Beach Head
Beach Head II
Blue Max
Castle Wolfenstein
Commando
Commando Libya Part I
Desert Fox
Eroticon
Falcon Patrol
Falcon Patrol II
Flyerfox
F 15 Strike Eagle
Friday the 13th
G.I. Joe I + II
Girls they want to have Fun
Green Baret
Hitler Diktator
Nice Demo
Paratrooper
Porno Dia Show
Protector II
Raid over Moscow
Rambo II
River Raid
S.D.I
Seafox/Seawolf
Sex Games
Silent Service
Skyfox
Soldier One
Speed Racer
Stalag I
Swedish Erotica
Stroker
Tank Attack
Teachbusters
Theatre Europe
1942 Trainer

Der Verlag behält sich vor, bei Softwareangeboten indizierte Spiele ersatzlos zu streichen

HIRES
Bewirkt eine erweiterte Pixelauflösung in der x-Achse, nämlich nun 640 Pixel. Die so erhöhte Auflösung erlaubt nur noch 16 Farben gleich 4 BitPlanes.

INTERLACE:
Eine Verdopplung der Pixelauflösung in der y-Achse, von 256 auf 512 Pixeln. Hier wird jedoch ein Trick angewendet. Die einzelnen Pixelzeilen werden nun nicht mehr alle 1/50 Sekunde auf dem Bildschirm erneuert, sondern in jedem Takt entweder nur die gradzahligen oder die ungradzahligen Reihen: Diese Vereinfachung führt, vor allem bei hohen Helligkeitskontrasten, zu Bildflackern.

SPRITES:
Dieses Flag muß gesetzt werden, wenn auf dem Screen Sprites genutzt werden sollen.

DUALPF:
Der "dual playfield mode". Hierbei können zwei unabhängige Sichtbereiche separat kontrolliert werden. Dies ist vor allem für Spiele brauchbar, wenn z.B ein Sichtbereich die Instrumententafel (playfield 1) und der andere die Sicht auf das eigentliche Spielgeschehen ist (playfield 2).

HAM:
Eine besondere Art der Farbverwaltung, welche alle 4096 Farben auf einmal ermöglicht. Dabei kann jedoch nicht jedes Pixel eine beliebige Farbe haben, da sich seine Farbe aus einer Differenz zu dem links davon liegenden Pixel bestimmt.

USHORT Type;
Normalerweise sollte hier <CUSTOMSCREEN> stehen, was bei einem selbstdefinierten Screen bedeutet, daß er sich Speicher entsprechend den gewünschten Ausmaßen reserviert. Man kann aber auch CUSTOMBITMAP setzen. Dann wird bei OpenScreen() keinBildschirmspeicher belegt, sondern ein vorher definierter Speicherbereich genutzt, der durch eine Struktur BitMap beschrieben wird, auf die dann ein noch folgendes Element zeigt.

struct TextAttr *Font;
Hier wird auf eine TextAttr-Struktur gezeigt, die für einen gewissen Font steht. Steht hier <NULL>, so wird der der Standardfont für den Screen und all seine Windows aktiv.

UBYTE *DefaultTitle;
Ein mit einem Null-Byte abgeschlossener String, der auf der Titlebar des Screens erscheint. Auch hier gilt für die Aztec-Benutzter, den Zeiger als UBYTE* bzw. STRPTR zu definieren.

struct Gadget *Gadgets;
Ein Element für die Zukunft. Daher darf hier nur <NULL> stehen.

struct BitMap *CustomBitMap;
Dieser Zeiger verweist auf eine Struktur, die einen eigenen Bildschirmspeicher beschreibt, ist also nur notwendig, wenn als Screen-Type <CUSTOMBITMAP> gesetzt wurde. Ansonsten steht hier <NULL>.

Mit diesen Informationen sollte es Ihnen möglich sein, die Darstellungsmöglichkeiten des Amiga ein wenig zu nutzen. Als kleines Beispiel dient das Listing screen&window.h. Es ist kein eigentliches Programm, sondern ein Headerfile, der mit der #include-Anweisung von dem eigentlichen Programm genutzt wird, so z.B von allen Programmen des Betriebssystemgrafikteils dieses Heftes.

screen&window.h beinhaltet das Öffnen der nötigen Libraries sowie das Initialisieren und Öffen eines Screens und eines Windows. Letzteres ist als Funktion zusammengefaßt und erspart somit viel Tipparbeit bei Programmen, in denen Screens und Windows benutzt werden. Eine zweite in screen&window.h enthaltene Funktion schließt das Window, den Screen und die Libraries wieder. Diese sollte vor dem Programmende immer aufgerufen werden, um den für Strukturen und Bildschirmspeicher reservierten Speicherplatz wieder freizugeben. Mit Hilfe einiger Betriebssystemaufrufe, welche in anderen Beiträgen dieses Heftes ausführlich erläutert sind, kann man nun Text und/oder Grafik auf ein Window bringen.

Nun stellt sich aber noch eine Frage: Wie kann ich dem unter Intuition laufenden Programm, also einem Programm, das Screens und Windows nutzt, etwas mitteilen, während es läuft?

Dies ist nicht so einfach, da nur die AmigaDos-Konsole, also das CLI-Window, mit den standardmäßigen Ein- und Ausgabefunktionen zusammenarbeitet.

Doch nicht verzagen, denn es gibt IDCMP.

Das heißt ausgeschrieben " Intuition Direct Communication Message Port". Der Name verspricht anscheinend die gewünschte "Communication" mit Intuition. Und das auch noch "Direct". Und einen "Message Port" gibt es auch noch. Um IDCMP zu nutzen, wird vorgegangen, wie im folgenden erläutert.

In der NewWindow-Struktur gibt es die Möglichkeit, IDCMP-Flags zu setzten. Das ist eine Grundvoraussetzung. Diese Flags definieren die verschiedensten Arten der Kontrolle aller Eingabegeräte. Was das Programm nun kontrollieren soll, legt man hier mit dem Setzen der Flags fest. Sind Flags gesetzt, so erhält man die Informationen über einen Message-Port, den UserPort des betreffenden Windows. Dort kommen die Nachrichten in Form einer Struktur IntuiMessage an, die aus folgenden Elementen besteht:

struct Message ExecMessage;
Dieses Element wird von Exec benutzt.

ULONG Class;
Hier erfährt man, welcher Art die Nachricht dieser Struktur ist. Hier steht eines der gesetzten IDCMP-Flags.

USHORT Code;
Manche Nachrichten hinterlassen hier Detailinformationen, z.B. bei der Tastaturkontrolle die gedrückte Taste oder deren ASCII-Zeichen.

USHORT Qualifier;
Wenn eine Sondertaste der Tastatur, wie z.B. <SHIFT>, <ALT>, <CTRL> etc., mit gedrückt wurde, steht hier ein entsprechendes <IEQUALIFIER>-Flag. Die genaue Bedeutung dieser Flags finden Sie im Headerfile <libraries/inputevent.h>.

APTR IAddress;
Ein Zeiger auf bestimmte Objekte, die mit der Nachricht in Verbindung stehen, wie beispielsweise Gadgets.

SHORT MouseX, MouseY;
Hier findet man die Mauskoordinaten relativ zum Window.

ULONG Seconds,Micros;
Die Systemzeit in Sekunden und Mikrosekunden.

struct Window *IDCMPWindow;

Ein Zeiger auf das Window, auf welches sich diese Nachricht bezieht.

Diese Struktur, oder auch eine sequentielle Folge davon, findet man auf dem UserPort des Windows. Mit Hilfe der Funktion GetMsg() kann man die IntuiMessages nacheinander von diesem Port holen und verarbeiten. Die letzte angekommene Nachricht findet man aber auch in der Window-Struktur selbst wieder. <Window->MessageKey> ist der entsprechende Zeiger darauf.

Bekommt man mehrere Nachrichten, so sollte man diese mit ReplyMsg() nach der Bearbeitung wieder freigeben, da sie sonst Speicher schlucken. Bleibt noch anzumerken, daß man die IDCMP-Flags auch nach dem Öffnen des Windows mit der Funktion ModifyIDCMP() ändern kann. Ihr wird als erstes der Zeiger auf ein bestehendes Window übergeben (man bekommt diesen Zeiger, wie gesagt, von der Funktion OpenWindow() zurück), und als zweites die Flags, die fortan für dieses Window gültig sein sollen.

Einige IDCMP-Flags wollen wir hier beschreiben. Alle abzuhandeln, würde dem Rahmen dieses Heftes, welches sich mit Grafik beschäftigt sprengen. Daher werden nur solche aufgeführt, die dazu dienen, Programme zu steuern.

CLOSEWINDOW

bezieht sich auf das Gadget, welches oben links im Window erscheint, wenn das Window-Flag <WINDOWCLOSE> gesetzt ist. Eine IntuiMessage wird diese Class haben, wenn die linke Maustaste über diesem Gadget gedrückt wird. Diese Methode wird in diesem Heft vor allem dazu verwendet, Programme zu beenden.

VANILLAKEY

Ist die Class <VANILLAKEY>, so findet man in Code das gerade auf der Tastatur gedrückte Zeichen als ASCII-Wert.

MOUSEMOVE

sorgt für Übertragung der Mausposition, doch muß dazu noch das Window-Flag <REPORTMOUSE> gesetzt sein.

MOUSEBUTTONS

Bei diesem Flag erhält man in Code Information über jedes Drücken und Loslassen der Maustasten. Das Code-Element nimmt dann einen der folgenden Werte an : SELECTDOWN, SELECTpUP, MENUDOWN, MENUUP.

Dabei bezieht sich <SELECT...> auf die linke, <MENU...> auf die rechte Taste der Maus.

Zum Umgang mit IDCMP finden Sie im Anschluß ein Listing.

```
=====================================================================
LISTING  VON :                                           Seiten ____
screen&window.h                 3750 rwed Today      18:34:31
=====================================================================

#ifndef INTUITION_INTUITIONBASE_H
#include <intuition/intuitionbase.h>
#endif

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *grafikscreen;
struct Window *grafikwindow;

/* * * * * * * * * * * * * * * * * * * * * *
 * DIE INITIALISIERUNG DES NEUEN SCREENS *
 * * * * * * * * * * * * * * * * * * * * * */

struct NewScreen screendata =
 {
   0,             /* Left Edge                  */
   0,             /* Top Edge                   */
   320,           /* Breite                     */
   256,           /* Hoehe                      */
   5,             /* Anzahl der BitPlanes       */
   0,             /* Detail Pen                 */
   1,             /* BlockPen                   */
   NULL,          /* Darstellungs Modus         */
   CUSTOMSCREEN,  /* Screen Typ                 */
   NULL,          /* Standard Font              */
   NULL,          /* Kein Titel auf der Titlebar */
   NULL,          /* keine Gadgets              */
   NULL           /* keine besondere BitMap     */
 };

/* * * * * * * * * * * * * * * * * * * * * *
 * DIE INITIALISIERUNG DES NEUEN WINDOWS *
 * * * * * * * * * * * * * * * * * * * * * */

struct NewWindow windowdata =
 {
   0,                         /* Linker Rand            */
   0,                         /* Oberer Rand            */
   320,                       /* Breite                 */
   255,                       /* Hoehe                  */
   0,                         /* DetailPen              */
   1,                         /* BlockPen               */
   CLOSEWINDOW,               /* IDCMP Flags            */
   ACTIVATE|WINDOWCLOSE,      /* Allgemeine Flags       */
   NULL,                      /* keine Gadgets          */
   NULL,                      /* kein eigenes Image     */
   (UBYTE *)"GRAFIKWINDOW",   /* Titel des Windows      */
   NULL,                      /* Hier wird noch kein    */
       /* Screen genannt da dieser erst geoeffnet wird*/
   NULL,                      /* ohne eigene BitMap     */
   NULL,                      /* Fuer die Window Limits*/
   NULL,                      /* werden die Anfangs -   */
   NULL,                      /* groessen uebernommen   */
   NULL,
   CUSTOMSCREEN,              /* Bezug auf einen        */
                              /* eigenen Screen         */
 };


make_display()
 {

/* * * * * * * * * * * * * *
 * OEFFNEN DER LIBRARIES *
 * * * * * * * * * * * * * */

  if(!IntuitionBase)
   {
    IntuitionBase = (struct IntuitionBase *)
                    OpenLibrary("intuition.library",0L);
    if(!IntuitionBase) /* Bei Fehler ... */
     {
      end_display();    /* ... Abbruch */
     }
   }

  if(!GfxBase)  /* Dasselbe fuer die Grafik */
   {
   GfxBase = (struct GfxBase *)
                    OpenLibrary("graphics.library",0L);
    if(!GfxBase)
     {
      end_display();
     }
   }

/* * * * * * * * * * * *
 * OEFFNEN DES SCREENS *
 * * * * * * * * * * * */

  grafikscreen = (struct Screen*)
                           OpenScreen(&screendata);
  if(!grafikscreen)  /* Bei Fehler ... */
   {
    end_display();   /* ... Abbruch    */
   }

/* Den neuen Screen der NewWindow Struktur zuweisen */
  windowdata.Screen = grafikscreen;

/* * * * * * * * * * * *
 * OEFFNEN DES WINDOWS *
 * * * * * * * * * * * */

  grafikwindow = (struct Window*)
                           OpenWindow(&windowdata);
  if(!grafikwindow)
   {
    end_display();
   }

  return(NULL);
 }

/* * * * * * * * * * * * * * * * * *
 * Schliessen und Freimachen, was*
 *geoefffnet und reserviert wurde*
 * * * * * * * * * * * * * * * * * */

end_display()
{
 if (grafikwindow) CloseWindow(grafikwindow);
 if (grafikscreen) CloseScreen(grafikscreen);
 if (GfxBase) CloseLibrary(GfxBase);
 if (IntuitionBase) CloseLibrary(IntuitionBase);
 return(NULL);
}

                   --- END OF FILE : screen&window.h ---
=====================================================================
```

```
================================================================
LISTING  VON :                                    Seiten ____
intu.c                     7464 rwed 16-Dec-87 14:18:11
==================================================== 3.4 a ======

/*************************************************************
 * Mit diesem Programm kann man sich die
 * Wirkungsweise der Window- und IDCMPFlags anschaulich
 * machen. Die Abfrageroutine, welche die IDCMP-
 * Messages konnntrolliert ld_t sich sicher leicht
 * aus dem Programm auskoppeln und f/r andere nutzen.
 *************************************************************/

#include <intuition/intuitionbase.h>

/* Die Window Dimensionen werden schon hier festgelegt
   da man sie fuer ein SUPERBITMAP-Window mehrmals
   braucht */
#define SBMW_BREITE 640L
#define SBMW_HOEHE  200L
#define SBMW_TIEFE    2L

struct IntuitionBase *IntuitionBase;
struct IntuiMessage *nachricht;
struct GfxBase *GfxBase;
struct Window *w;
struct BitMap bm;
struct RastPort *rp;

/* * * * * * * * * * * * * * * * * * * * *
 * DIE INITIALISIERUNG DES NEUEN WINDOWS *
 * * * * * * * * * * * * * * * * * * * * */

struct NewWindow nw =
 {
   0,                          /* Linker Rand          */
   0,                          /* Oberer Rand          */
   SBMW_BREITE,                /* Breite               */
   SBMW_HOEHE,                 /* Hvhe                 */
   0,                          /* DetailPen            */
   1,                          /* BlockPen             */
   NULL,                       /* IDCMP Flags          */
   NULL,                       /* Allgemeine Flags     */
   NULL,                       /* keine Gadgets        */
   NULL,                       /* kein eigenes Image   */
   (STRPTR)"FLAGTEST",         /* Titel des Windows    */
   NULL,                       /* Bezug auf WBScreen   */
   NULL,                       /* noch keine BitMap    */
   10,                         /* Die                  */
   10,                         /* Window               */
   NULL,                       /* Limitationen         */
   NULL,
   WBENCHSCREEN,               /* Bezug auf den        */
                               /* WorkBenchScreen      */
 };

main()
 {
  ULONG i,input = NULL;
  SHORT counter;
  STRPTR key;

  /* * * * * * * * * * * * *
   * OEFFNEN DER LIBRARIES *
   * * * * * * * * * * * * */

  if(!IntuitionBase)
   {
    IntuitionBase = (struct IntuitionBase *)
                    OpenLibrary("intuition.library",0L);
    if(!IntuitionBase) /* Bei Fehler ... */
     {
      ende();   /* ... Abbruch */
     }
   }

  if(!GfxBase)  /* Dasselbe f/r die Grafik */
   {
    GfxBase = (struct GfxBase *)
              OpenLibrary("graphics.library",0L);
    if(!GfxBase)
     {
      ende();
     }
   }

  printf("\nWieviel IDCMP-Messages sollen ");
  printf("abgearbeitet werden\n");
  printf("bevor das Programm abbricht ?:");
  scanf("%d",&counter);

 /* Die Eingabe der Wimdowflags in deren effektiven
    Werten. Es ist wohl nicht nvtig alle printf()
    Zeilen einzugeben, da man die Werte au h  is
    dem Heft lesen kann */

  printf("\n\n\n");
  printf("WINDOWFLAGS SETZEN\n\n");
  printf("WINDOWSIZING = 1      WINDOWDRAG      = 2\n");
  printf("WINDOWDEPTH  = 4      WINDOWCLOSE     = 8\n");
  printf("SIZEBRIGHT   = 16     SIZEBBOTTOM     = 32\n");
  printf("SMARTREFRESH = 0      SIMPLE_REFRESH  = 64\n");
  printf("SUPER_BITMAP = 128    BACKDROP        = 256\n");
  printf("REPORTMOUSE  = 512    GIMMEZEROZERO   = 1024\n");
  printf("BORDERLESS   = 2048   ACTIVATE        = 4096\n");
  printf("RMBTRAP      = 65536  NOCAREREFRESH   = 131072\n");
  printf("\nWerte der gew/nschten Flags nacheinander");
  printf(" eingeben\n und nach dem letzten die 3\n");

  while(input != 3)
   {
    nw.Flags += input;
    printf("\nMomentaner Flagwert :%ld  Dazu ?:",nw.Flags);
    scanf("%ld",&input);
   }

  if(nw.Flags & 128) /* Wenn SUPER_BITMAP gesetzt */
   {
    printf("\n ein SUPER_BITMAP window\n");

    /* * * * * * * * * * * * *
     * ERZEUGEN DER BITMAPS  *
     * * * * * * * * * * * * */

    InitBitMap(&bm,SBMW_TIEFE,SBMW_BREITE,SBMW_HOEHE);
    for(i = 0;i < SBMW_TIEFE;i++)
     {
      bm.Planes[i] = (PLANEPTR)
                     AllocRaster(SBMW_BREITE,SBMW_HOEHE);

      if(!bm.Planes[i])
       {
        printf("\nKein Speicher fuer BitPlanes !\n");
        ende();
       }

     /* Die Planes lvschen von eventuellen Datenresten
        mit der Routine BltClear() */

      BltClear(bm.Planes[i],
               ((SBMW_BREITE * SBMW_HOEHE)/8),NULL);

     }

    /* Und den BitMap Zeiger in die NewWindow Struktur
       einf/gen */

    nw.BitMap = (struct BitMap *)&bm;

   }

  /* Der selbe Vorgang f/r die IDCMPFlags */

  printf("\n\n\n\n\nIDCMP-FLAGS SETZEN :\n\n");
  printf("MOUSEBUTTONS = 8     MOUSEMOVE  = 16\n");
  printf("CLOSEWINDOW  = 512   VANILLAKEY = 2097152\n");
  printf("\nFuktioniert wie die Windowflageingabe\n");


  input = NULL;
  while(input != 3)
   {
    nw.IDCMPFlags += input;
    printf("\nMomentaner IDCMP ");
    printf("Wert :%ld  Dazu ?:",nw.IDCMPFlags);
    scanf("%ld",&input);
   }

  /* * * * * * * * * * *
   * OEFFNEN DES WINDOWS *
   * * * * * * * * * * */

  w = (struct Window *)OpenWindow(&nw);
  if(!w)
   {
    ende();
   }

  /* Bestimmung des RastPorts */
  rp = w->RPort;


  /* Ein wenig Grafik auf das Window */
  for(i = 0;i < 4;i++)
   {
    SetAPen(rp,i);
    Move(rp,20L,10L*i+20L);
    Draw(rp,300L,10L*i+20L);
   }

 /* counter wird bei jedem Schleifendurchlauf
    heruntergezdhlt, dadurch entsteht eine Abbruch-
    mvglichkeit fuer das Programm */

  while(counter--)
   {
    /* Beginn der IDCMP Abfrage */

    /* Warten auf Ankunft einer Message auf dem
       UserPort eines Windows */
    Wait(1L<< w->UserPort->mp_SigBit);


    /* Dann die Message mit GetMsg() vom Port holen */
    while(nachricht = (struct IntuiMessage*)
                      GetMsg(w->UserPort))
     {
      /* Loeschen einer Textzeile im Window */
      Move(rp,50L,50L);
      Text(rp,"                    ",20L);

      /* Die Art der Nachricht entspricht dem IDCMPFlag
         in <Class>, daher wird nun dieses untersucht */
      switch(nachricht->Class)
       {
        case MOUSEBUTTONS :
                        Move(rp,50L,50L);
                        Text(rp,"Mausknopf",9L);
                        break;

        case MOUSEMOVE :
                        Move(rp,50L,50L);
                        Text(rp,"Maus bewegt",11L);
                        break;

        case CLOSEWINDOW :
                        Move(rp,50L,50L);
                        Text(rp,"Close Gadget",12L);
                        Delay(50L);
                        break;
```

```
        case VANILLAKEY :
                         Move(rp,50L,50L);
                         Text(rp,"Taste gedruckt",12L);
                         Move(rp,180L,50L);
                     /* Das Code Element ist ein WORD aus
                        dem nur das untere Byte den ASCII-
                        Wert des Zeichens enthält, daher
                        wird der Pointer hochgezählt */
                        key = (STRPTR)&nachricht->Code;
                        key++;
                        Text(rp,key,1L);
                        break;
      }

    /* Wenn die Nachricht abgearbeitet ist wird sie
       freigegeben */
     ReplyMsg(nachricht);
    }

   }

   Move(rp,20L,20L);
   Text(rp,"Das war die letzte Nachricht ! Abbruch"
              ,38L);

     ende();

   }


 ende()
 {
  LONG i;

 /* Window und Planes schliessen */
  if (w) CloseWindow(w);
  for(i = 0;i < SBMW_TIEFE;i++)
   {
    if(bm.Planes[i]) FreeRaster(bm.Planes[i],
                          SBMW_BREITE,SBMW_HOEHE);
   }

 /* Librarys schliessen */
  if (GfxBase) CloseLibrary(GfxBase);
  if (IntuitionBase) CloseLibrary(IntuitionBase);
  exit(1L);
 }

                  --- END OF FILE : intu.c ---
==================================================================
```

VON WOLF DIETRICH

GRAFIK

MIT DEM BETRIEBSSYSTEM

Ohne Zweifel stellt der Amiga Programmierern hervorragende Unterstützung betreffs implementierter Funktionen zur Verfügung. Ausgerüstet mit einer Vielzahl grafischer Funktionen, welche in ein komplexes Netz von Datenstrukturen eingebunden sind, sollte er das Herz eines jeden grafikinteressierten Anwenders im Sturm erobern. Allerdings benötigt man bei einer solchen Komplexität auch entsprechend viel an Informationen, um die Funktionen, die zur Verfügung stehen, auch anwenden zu können.

Für den C-Programmierer war das bislang gleichzusetzen mit der Anschaffung einer kleinen Bibliothek. Um dies zu umgehen, sollen in diesem Artikel die wichtigsten grafischen Funktionen aufgeführt und anhand einfacher Beispiele erläutert werden. Dies soll natürlich nicht in eine umfassende und tiefgreifende Diskussion des Amiga-Betriebssystems ausarten, da dafür auch einiges an Hardwarewissen vorausgesetzt werden müßte und es auf einigen wenigen Seiten nicht abgehandelt werden könnte; vielmehr sollen leicht verständliche Beispiele einen schnellen Einstieg in die Anwendung der Betriebssystemroutinen erleichtern, wofür allerdings ein gewisses Grundwissen über die Programmierung in C und vor allem über Zeiger und Strukturen vorausgesetzt wird.

Vielleicht regt dieser Bericht aber auch den einen oder anderen Nicht-C-Kundigen dazu an, sich doch einmal mit dieser hocheffizienten und schnellen Sprache zu beschäftigen, denn so kompliziert, wie es auf den ersten Blick den Anschein hat, ist C längst nicht. Doch genug jetzt der einleitenden Worte, es geht ans Eingemachte.

Die AreaFill-Funktionen

In dieser Gruppe kann man mehrere Funktionen zusammenfassen, die sich durch einfache Handhabung und hohe Effizienz auszeichnen. Im einzelnen handelt es sich um die Funktionen AreaCircle(), AreaEllipse(), RectFill(), Flood() und die Kombination aus AreaMove() und AreaDraw(). Ihnen allen ist gemeinsam, das sie mit kurzen Aufrufen komplexe Ergebnisse erbringen. Allerdings sind diese Befehle an Datenstrukturen gebunden, die der Programmierer vor Benutzung dieser Funktionen initialisieren muß, denn sonst beginnt der Amiga zu meditieren. Um sich den Grund dafür zu veranschaulichen, muß man sich vor Augen halten, was der Amiga bei Aufruf dieser Funktionen macht. Im Prinzip erscheint dies einfach, denn er zeichnet lediglich gefüllte Flächen, Kreise oder Ellipsen auf den Bildschirm. Doch ganz so einfach, wie es scheint, geht dies nicht vonstatten. Bevor das Betriebssystem die Flächen zeichnen kann, muß das die Routinen aufrufende Programm nämlich erst einmal alle Zeichenkoordinaten übergeben haben, und diese müssen irgendwo festgehalten werden. Weiterhin kann es auch nicht sofort in die einzelnen Bitmaps des Screens zeichnen, sondern muß die Fülloperationen auf einer gesonderten Rasterbitmap durchführen und den Inhalt dieser Rasterbitmap dann in die einzelnen Bitmaps des Screens kopieren, je nach der aktuellen Zeichenfarbe in Form von gesetzten oder ungesetzten Bits (für jene Leser, die mit dem Begriff Bitmap nicht vertraut sind: Es handelt sich dabei um Speicherblöcke, in denen die Bildschirmdaten liegen, und von ihrer Anzahl ist die Zahl der Farben auf dem Screen abhängig; letztere ergibt sich aus der Zahl der möglichen Bitkombinationen für ein Pixel, die bei fünf Bitplanes, wie im Includefile screen&window.h, 2 hoch 5 , also 32, beträgt).Um also das Betriebssystem zufriedenzustellen, muß vor Aufruf der oben genannten Funktionen Speicherplatz freigemacht

werden, und das System muß mitgeteilt bekommen, wo es diesen findet.
All diese Initialisierungen und Zuweisungen gehen über fest definierte Strukturen vonstatten, die nun im einzelnen erläutert werden sollen.

Die RastPort-Struktur

Diese Struktur nimmt im Betriebssystem eine bedeutende Stellung ein, denn sie kontrolliert alle Festlegungen einer Bitmap oder auch eines Screens oder eines Windows; sie verweist auf andere wichtige Strukturen, enthält Daten für verschiedene Zeichenmodi und Zeiger auf reservierte Datenbereiche, die zur Ausführung diverser Routinen (wie auch der hier besprochenen) nötig sind. Sie erweist sich für den Anwender in vielen Belangen als "erste Adresse". Man kann eine RastPort-Struktur von Hand eröffnen und initialisieren, aber auch jeder Screen und jedes Window, die über Intuition eröffnet wurden, enthalten eine spezifische RastPort-Struktur. Da die Beispielsprogramme mit einem Screen und einem daraufliegenden Window arbeiten, benötigt man, um auf das Window zeichnen zu können, die Adresse seiner RastPort-Struktur. Die Window-Struktur selbst zeigt auf diese, sie ist in der Window-Struktur als <struct RastPort* RPort> festgelegt. Um diesen Begriff nicht ständig gebrauchen zu müssen, kann man einen weiteren RastPort-Struktur-Pointer global definieren und der Window-RastPort-Struktur zuweisen:

```
struct RastPort* rasp;
struct Window* grafikwindow;

main()
{
...
 rasp = grafikwindow->RPort;
...
}
```

Wie Sie später noch sehen werden, benötigen fast alle der Zeichenfunktionen des Betriebssystems die Adresse des RastPorts, denn die RastPort-Struktur kontrolliert die Displaydefinitionen wie auch die Displayhardware.

Die AreaInfo-Struktur

Diese Struktur stellt dem Betriebssystem Information über, oder besser, Zeiger auf reservierte Speicherbereiche zur Verfügung. In diesen Bereichen werden von den verschiedenen AreaFill-Operationen Punktlisten erstellt, die dann abgearbeitet werden, denn die Ausführung einer AreaFill-Operation geht Schritt für Schritt vor sich. Diese Struktur muß vor Benutzung initialisiert werden, und zwar mit folgendem Aufruf:

```
InitArea(&a_info,&areabuf[0],wert);
```

Was bedeuten die einzelnen Parameter dieses Aufrufs? Nun, <a_info>ist eine Struktur vom Typ AreaInfo, oder genauer die Struktur, die initialisiert wird. Die Variable <areabuf> ist ein Array von UnsignedWords(UWORDS) ein Speicherblock, in dem später die Koordinaten der zu zeichnenden Punkte als x,y-Paare abgelegt werden. Es muß als UWORD-Array festgelegt werden, da die Startadresse von <areabuf> auf einer geraden Adresse liegen muß; andernfalls könnte es von Seiten des Amiga zu unangenehmen Reaktionen kommen. Mit <wert> wird dem Funktionsaufruf mitgeteilt, wieviele Punkteinträge in die Liste maximal vorgenommen werden können. Diese Variable korrespondiert direkt mit <areabuf>: Wenn Sie zum Beispiel <wert> auf 100 setzen, was bedeutet, daß maximal 100 Punkte zwischen zwei AreaEnd()-Aufrufen gezeichnet werden können, so müssen Sie, da in <areabuf> 5 Bytes pro Punkt reserviert sein müssen, <areabuf> als <UWORD areabuf [250] > definieren (250 UWORDS entsprechen im Speicherplatz 500 Bytes). Achten Sie stets darauf, genügend Speicherplatz für die Punktlisten zur Verfügung zu stellen, denn sonst werden die von Ihnen aufgerufenen Funktionen nicht ordnungsgemäß ausgeführt.

Die TmpRas-Struktur

Diese Strukur hält die Adresse einer Zeichenbitmap und die Größenangabe selbiger für das System bereit. Diese Zeichenbitmap benötigen sämtliche AreaFill-Operationen, denn in ihr wird jede zu zeichnende Fläche zuerst gezeichnet und danach in die Bitmap des Screens kopiert. Dieses Prinzip muß angewandt werden, da für jede Fläche eine freie Bitmap notwendig ist, denn eine Fläche wird zuerst umrandet und dann vom Blitter über logische Operationen gefüllt. Dies läßt sich nicht in einer Bitmap erledigen, in der schon Bits gesetzt sind, also irgendetwas gezeichnet ist, da der Blitter dann Flächenränder erkennen würde, wo gar keine sind. Eine solche Zeichenbitmap muß man sich selbst definieren; sie wird als <BYTE*> festgelegt.
Es ist außerdem notwendig, den erforderlichen Speicherplatz zu reservieren und dessen Startadresse dem Byte-Pointer zuzuweisen. Dies geschieht über die Funktion AllocRaster():

```
BYTE* area_plane;

main()
{
...
                area_plane =
AllocRaster(Breite,Höhe);
...
}
```

AllocRaster() benötigt als Parameter die Breite und die Höhe der Zeichenbitmap in Pixeln. Im Normalfall entsprechen diese Werte der Breite und der Höhe Ihres Windows. Vor dem Verlassen des Programms muß dieses den reservierten Speicherplatz wieder an das System zurückgeben; dies geschieht mit dem Aufruf:

```
FreeRaster(area_plane,Breite,Höhe);
```

Um die TmpRas-Struktur zu initialisieren, verwenden Sie den Aufruf InitTmpRas():

```
struct TmpRas t_ras

main()
{
...
InitTmpRas(&t_ras,area_plane,RASSIZE(Breite,Höhe));
...
}
```

RASSIZE() ist ein Systemmacro, das die für das Raster benötigte Größe berechnet. Ihm müssen die gleichen Werte wie dem AllocRaster()-Aufruf übergeben werden. (Anmerkung: In der 3.2-Version des Aztec-CCompilers arbeitet der Aufruf AllocRaster() nicht plangemäß und führt zu Systemabstürzen. Wenn Sie mit diesem Compiler arbeiten, so können Sie AllocRaster() durch den Aufruf <area_plane = AllocMem(wert,MEMF_CHIP)> ersetzen, wobei sich <wert> aus Breite mal Höhe geteilt durch acht errechnet. Um so reservierten Speicherplatz an das System zurückzugeben, verwenden Sie den Aufruf <FreeMem(area_plane,wert>). Die Initialisierung der aufgeführten Strukturen ist eine der Grundvoraussetzungen für die Anwendung der AreaFill-Operationen. Zusätzlich müssen Sie aber die verschiedenen Strukturen noch miteinander verknüpfen, wozu folgende zwei Zuweisungen durchzuführen sind:

```
rasp->AreaInfo = &a_info;
rasp->TmpRas = &t_ras;
```

So, damit haben Sie schon das Wichtigste über die Strukturen für die AreaFill-Operationen erfahren. Mit dieser Grundlage können Sie alle Funktionen dieser Gruppe anwenden. Bevor diese selbst erläutert werden, können Sie ja schon einmal einen Blick auf das Beispielslisting area_graphics.c werfen. Hierin werden alle oben diskutierten Aufrufe angewandt, und auch die Variablennamen sind die gleichen wie in den Beispielen, wenn nicht bereits feste numerische Werte übergeben werden. Bitte beachten Sie, daß die Funktionen InitArea() bis InitTmpRas() in der Reihenfolge ausgeführt werden müssen, wie dies in area_graphics.c getan wird, da der Amiga sonst das berühmte AMIGA_FIREWORKS_DISPLAY oder aber gar nichts mehr zeigt. Es wird Ihnen vielleicht noch auffallen, daß sämtliche Zahlenwerte, die direkt an Funktionen übergeben werden, mit einem großen "L" versehen sind. Dies ist in Aztec-C notwendig, da der Aztec-Compiler 16-Bit-Integerwerte zur Verfügung stellt, während das Betriebssystem bei den meisten Aufrufen als Übergabeparameter 32-Bit-Werte erwartet; deshalb werden alle Werte mit dem Cast Operator "L" in Longs umgewandelt, die auch beim Aztec 32 Bit lang sind. Diesem Zweck dient auch der vorangestellte Cast Operator der Form (long), welcher bei Variablennamen Verwendung findet. Sollten im Programm mysteriöse Fehlfunktionen auftreten, so überprüfen Sie auch, ob Sie kein großes "L" oder ein (long) vergessen haben. Für Lattice-Anwender stellt sich dieses Problem allerdings erst gar nicht.

Nun aber zu den Funktionen. Als erstes sei hier die "komplizierteste" aufgeführt, die Kombination von AreaMove() und AreaDraw(). Diese dient zum Füllen beliebiger Vielecke, wobei der Startpunkt des Vielecks mit einem AreaMove()-Aufruf festgelegt wird:

```
AreaMove(rasp,xposition1,yposition1);
```

Wie Sie sehen, wird hier wieder der RastPort benötigt, aber das nur nebenbei. Die Variablen <xposition1> und <yposition1> beziehen sich auf die Bildschirmkoordinaten, wobei die linke obere Ecke den 0,0-Punkt darstellt. Haben Sie den ersten Punkt festgelegt, so können Sie weitere Punkte definieren, wobei jeder stets mit dem Vorhergehenden verbunden wird.

Dies geschieht über die Funktion AreaDraw():

```
AreaDraw(rasp,xposition2,yposition2);
AreaDraw(rasp,xposition3,yposition3);
....
```

In der Anzahl der Punkte sind Sie nur durch Ihre eigene Angabe im InitArea()-Aufruf (und irgendwann durch den Speicherplatz des Amiga)limitiert. Haben Sie schließlich alle Punkte gesetzt, so lassen Sie die Fläche mit dem folgendem Aufruf zeichnen:

```
AreaEnd(rasp);
```

Diese Funktion weist das System an, die Fläche zu schließen (Ihr Start-und Endpunkt müssen nicht die gleichen sein) und zu zeichnen. Das Ergebnis eines solchen Aufrufs (oder besser, mehrerer dieser Sorte)können Sie im Programm area_graphics bewundern.

Ein weiterer Aufruf der AreaFill-Operationen ist der Befehl AreaEllipse(). Er dient, wie schon der Name sagt, zum Zeichnen einer gefüllten Ellipse und benötigt ein paar Parameter mehr als AreaMove() oder AreaDraw():

```
AreaEllipse(rasp,xposition,yposition,
hor_radius,ver_radius);
AreaEnd(rasp);
```

Die ersten drei Parameter entsprechen denen in den vorangegangenen Aufrufen; der horizontale und der vertikale Radius werden in Pixeln angegeben. Auch AreaEllipse() benötigt einen AreaEnd()-Aufruf, andernfalls wird die Ellipse nicht gezeichnet.

Der letzte Befehl in dieser Reihe ist der AreaCircle()-Befehl, mit dem ein gefüllter Kreis gezeichnet wird. Er entspricht dem AreaEllipse()-Befehl und weist nur eine kleine Änderung auf:

```
AreaCircle(rasp,xposition,yposition,radius)
AreaEnd(rasp);
```

Logischerweise benötigt ein Kreis nur eine Radius-Angabe. Ansonsten bleibt alles beim Alten.

Zwei weitere Befehle, die allerdings nicht mit Area- beginnen, benötigen ebenfalls die AreaInfo- und TmpRas-Struktur, um ausgeführt werden zu können: RectFill() und Flood(). RectFill() zeichnet ein geradliniges ge-

fülltes Rechteck auf den Bildschirm. Diese Funktion wird folgendermaßen aufgerufen:

```
RectFill(rasp,xmin,ymin,
xmax,ymax);
```

<xmin> und <ymin> geben die Koordinaten der linken oberen, <xmax> und <ymax> die der rechten unteren Ecke des Rechtecks an. Dieser Befehl wird sofort ausgeführt, ebenso wie Flood(), welcher zum Füllen mit der gewählten Farbe an einem beliebigen Punkt dient und folgendermaßen benutzt wird:

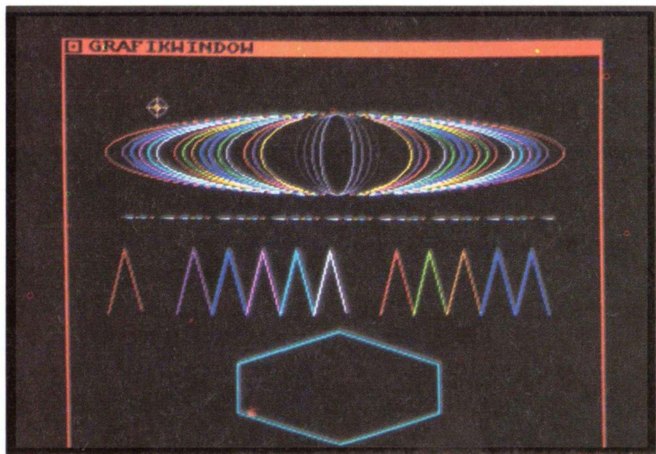
```
Flood(rasp,modus,xposition,yposition)
```

<xposition> und <yposition> geben die Bildschirmkoordinate an, auf welcher Flood() mit der Ausführung beginnt. Als <modus> stehen dem Anwender zwei Möglichkeiten zur Verfügung: 0 weist Flood() an, alle Pixel zu füllen, die nicht die Farbe des aktuellen AreaOutlinePens (AOlPen) haben (mehr dazu weiter unten),während bei 1 alle Pixel gefüllt werden, die die gleiche Farbe haben wie das Startpixel.

Noch ein paar Worte zum Beispielsprogramm area_graphics: Den AreaCircle()-Aufruf werden Sie vergeblich suchen, da in der 3.4-Version des Aztec-Compilers, mit dem das Programm erstellt wurde, dieser Befehl nicht implementiert ist, was allerdings nicht schlimm ist, da man ihn ja mit dem AreaEllipse()-Befehl "simulieren" kann. Bedeutsamer ist da die kleine for(;;)-Schleife, die den <areabuf> leert. Diese wurde eingefügt, da ohne sie beim AreaEllipse()-Aufruf scheinbar noch alteFlächenkoordinaten (oder "Reste" davon) im Array standen und beim Zeichnen der Ellipsen zu unschönen Ecken führten. Dies sollte zwar eigentlich nicht der Fall sein, aber wenn das Betriebssystem wieder einmal launisch ist, so garantiert diese kleine Schleife in jedem Fall, daß sauberes Zeichnen gewährleistet ist. Zur Farbwahl beim Füllen der Flächen dient der Befehl SetAPen(), welcher noch genauer erläutert wird. Um area_graphics.c zu compilieren, müssen Sie den Includefile screen&window.h, welcher auch in diesem Heft abgedruckt ist, miteinbinden, denn er versorgt Ihr Programm mit den nötigen globalen Variablen, und erleichtert Ihnen die Arbeit, da Sie nur noch die Funktion make_display() aufrufen müssen, um einen Screen und ein Window zu eröffnen. Dies gilt für alle abgedruckten Beispielsprogramme. Verlassen können Sie area_graphics, wie auch die anderen Beispielsprogramme, mit einem Mausklick auf das Close-Gadget des Windows.



Das Ergebnis von area_graphics



Linien, Punkte und Elipsen: simple_graphics

Ungefülltes Zeichnen

Neben den AreaFill-Funktionen, die alle gefüllte Flächen erzeugen, gibt es noch eine Reihe von "normalen" Funktionen für das Zeichnen von Linien, Punkten und Kreisen. Sie haben eine sehr einfache Handhabung gemeinsam, denn außer einem RastPort, in den gezeichnet werden soll, benötigen sie keine weiteren vorher initialisierten Strukturen. Da der RastPort vom Window (oder auch Screen) zur Verfügung gestellt wird, kann man also gleich mit dem Aufrufen der Funktionen loslegen. Als Beispielsprogramm für diese Befehle dient das Listing simple_graphics.c. Da es dem Listing area_graphics.c teilweise gleicht, sind darin nur neue Funktionen kommentiert. Nach der Initialisierung von <rasp> geht es hier dann auch gleich zur Sache, und zwar mit dem Befehl Move(). Dieser dient dazu, den Grafikcursor an eine beliebige Stelle auf dem Bildschirm zu manövrieren. Er benötigt als Übergabeparameter nur den RastPort und die x- und y-Koordinaten:

```
Move(rasp,xposition,yposition);
```

Move() muß vor Anwendung des Befehls Draw(), welcher eine Linie zieht, aufgerufen werden, denn sonst wird als Ausgangsposition für Draw() die letzte Position des Grafikcursors gewählt (zum Beispiel der Mittelpunkt eines Kreises, den Sie vorher haben zeichnen lassen). Move() positioniert den Cursor auch für den Einsatz der Funktion Text(), doch das gehört zum Thema Fonts. Ruft man, wie im vorliegenden Beispiel, mehrere

Draw()-Befehle hintereinander auf, so hängen die erzeugten Linien aneinander. Draw() benötigt die gleichen Parameter wie Move():

```
Draw(rasp,xposition,yposition);
```

Äquivalent zu den Aufrufen AreaEllipse() und AreaCircle() sind die Befehle Ellipse() und Circle(); der einzige Unterschied liegt darin, daß letztere eben nicht füllen. Auch die Übergabeparameter sind die gleichen:

```
Ellipse(rasp,xposition,yposition,
hor_radius,ver_radius);
Circle(rasp,xposition,yposition,
radius);
```

Auch der Circle()-Befehl ist im Programm simple_graphics ohne Beispiel, da er in der Aztec 3.4-Version nicht implementiert ist. Wichtiger als dieser Befehl ist allerdings auch der Aufruf PolyDraw(), welcher eine beliebige Anzahl miteinander verbundener Linien auf einmal zeichnet. Man muß ihm zu diesem Zweck allerdings ein Punktarray zur Verfügung stellen, das er abarbeiten kann. PolyDraw() wird folgendermaßen aufgerufen:

```
PolyDraw(rasp,wert,&pkt_liste[0]);
```

<wert> ist in diesem Beispiel die Anzahl der Wertepaare, die in der Punktliste stehen und jeweils die x- und y-Koordinaten eines Punkts angeben. Dieses Punktarray muß als Array von SHORT-Werten definiert werden, da sonst die Funktion PolyDraw() nicht das gewünschte Ergebnis bringt. Übergeben wird PolyDraw() die Adresse des ersten Elements der Punktliste, welche man über den Adressoperator <&> erhält.

Um auf einzelne Pixel Einfluß nehmen zu können, existiert der Befehl WritePixel(). Sein Aufruf wird in der folgenden Art und Weise getätigt:

```
WritePixel(rasp,xposition,yposition);
```

Auf diesen Aufruf hin wird ein mit den Parametern <xposition> und <yposition> beschriebenes Pixel in der aktuellen Zeichenfarbe gesetzt.

Um den Farbwert eines beliebigen Pixels herauszufinden, kann man den Befehl ReadPixel() einsetzen:

```
farbe= ReadPixel(rasp,
xposition,yposition);
```

<farbe> muß eine 32-Bit-Integervariable, in Lattice-C also vom Typ INT und in Aztec-C vom Typ LONG sein, und das Ergebnis ist der Index der gesetzten Farbe, kann bei einem Screen mit fünf Bitplanes also von 0 bis 31 reichen. Eine sinnvolle Anwendung dieser Funktion wäre beispielsweise, in einem Malprogramm, um aus einer dargestellten Farbpalette die Zeichenfarbe auszuwählen, auf einen Mausklick hin die Position des Mauszeigers abzufragen und mit dessen Koordinaten einen ReadPixel()-Aufruf durchzuführen.

Dies war schon das Wesentliche der Zeichenoperationen, und wie Sie im Listing simple_graphics.c erkennen werden, ist ihre Anwendung ausgesprochen einfach (woher das Programm ja auch seinen Namen hat).

So stehen nun die Farb- und Musteroperationen an, denen man noch etwas Aufmerksamkeit schenken sollte, denn sie ermöglichen mannig-faltige Arten der Beeinflussung dessen, was beim Aufruf der vorher diskutierten Funktionen auf dem Bildschirm erscheint.

Der Modus macht's

Wie Ihnen sicher schon aufgefallen ist, sollte man, um gezielte Ergebnisse zu erhalten, Einfluß auf die Farbsetzung eines Programms nehmen. Die vorangegangenen Beispiele verwenden die normale Farbliste eines neu eröffneten Screens. Damit sollte man sich allerdings, bei den gegebenen Möglichkeiten, nicht abfinden, und so stellt sich die Frage:Wie kriege ich meine Wunschfarben in mein Programm? Dies ist eine sehr einfache Angelegenheit, denn Sie benötigen lediglich eine Farbpalette und die Adresse der Struktur ViewPort, wie sie in der Struktur eines jeden Screens zu finden ist. Letztere erhalten Sie, indem Sie in Ihrem Programm folgendes definieren:

```
struct ViewPort* viep;
struct Screen* grafikscreen;
main();
{
...
viep = &grafikscreen->ViewPort;
...
}
```

Die Farbliste allerdings müssen Sie selbst erstellen; sie wird als Array von UWORD-Werten definiert. Enthalten muß sie so viele UWORDs, wie Ihr Screen Farben zuläßt, also 2, 4, 8, 16 oder 32. Diese UWORDs legt man am besten in Form von Hex-Werten an, da in einem Hex-Wert des Formats <0xnnn> die drei hinteren Stellen jeweils einen Farbanteil beeinflussen, der Rot, Grün oder Blau repräsentiert. So stellen die Werte <0xf00> reines Rot, <0x0f0> reines Grün und <0x00f> reines Blau dar. Aus den verschiedenen Kombinationen ergeben sich dann die 4096 verschiedenen Farben des Amiga. Als Beispiel hierzu dient die Farbliste im Listing colors&patterns.c. Haben Sie es hinter sich gebracht, die verschiedenen Farben zu definieren (gar nicht so einfach, sich interessante Kombinationen einfallen zu lassen), so müssen Sie im Programm die Farbliste in den Screen einladen.

Dies geschieht mit folgendem Aufruf:

```
LoadRGB4(viep,&farbliste[0],wert);
```

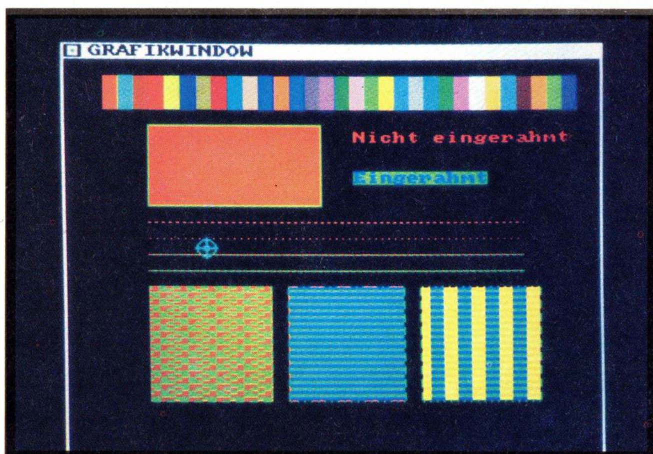
<wert> gibt hier die Anzahl der Farben, die geladen werden sollen an, die in jedem Fall der Anzahl der Werte in der Farbliste entsprechen sollte. Weiterhin benötigt LoadRGB4() die Adresse des Viewports des betreffenden Screens und die Adresse des ersten Elements in der Farbliste. Nach diesem Aufruf stehen die von Ihnen gewählten Farben zur Verfügung.

Will man einzelne Farben verändern, so ist es nicht nötig, gleich eine ganz neue Farbpalette zu laden. Vielmehr steht für diesen Zweck die Funktion SetRGB4() zur Verfügung, welche folgende Übergabeparameter benötigt:

```
SetRGB4(viep,index,rotanteil,
grünanteil,blauanteil);
```

<index> gibt die Farbe an, welche

beeinflußt werden soll. Im vorliegenden Beispielsprogramm mit 32 Farben reicht <index> von 0 bis 31. <rotanteil>, <grünanteil> und <blauanteil> reichen jeweils von 0 bis 15. Sie entsprechen den letzten drei Stellen der Werte der Farbliste.
Nun wollen Sie aber sicher nicht nur Farben definieren, sondern diese auch gezielt einsetzen. Hierzu müssen Sie auf die drei Zeichenfarben Einfluß nehmen, die im System definiert sind, als da wären: die primäreZeichenfarbe (APen), die sekundäre Zeichenfarbe (BPen) und die AreaOutline-Farbe (AOlPen), welche zum Umranden von Flächen dient.
Die Festlegung dieser Farben geschieht in drei sehr ähnlichen Aufrufen:



Farben und Muster ganz nach Wunsch: colors & patterns

```
SetAPen(rasp,index);
SetBPen(rasp,index);
SetOPen(rasp,index);
```

<index> gibt auch bei diesen Befehlen die Farbe an, die verwendet werden soll. Jede der drei Zeichenfarben erfüllt spezifische Aufgaben.
APen dient allgemein zum Zeichnen und Füllen, während BPen nur in bestimmten Zeichenmodi erscheint; wählen Sie zum Beispiel gepunktete Linien, so haben die Punkte abwechselnd die Farben des APen und des BPen, und auch die Flächenfüllmuster verwenden teilweise beide Pens.
Auch beim Aufruf der Text()-Funktion spielt der BPen eine Rolle, denn er bestimmt die Farbe, in der die Schrift unterlegt wird. Der AOlPen dient zum Umranden von Flächen; bei Anwendung der verschiedenen AreaFill-Funktionen werden die gezeichneten Flächen mit der über SetOPen() festgelegten Farbe eingerahmt. Auch hierfür finden sich einige Beispiele in colors&patterns.c.
Um die kurz zuvor erwähnten gepunkteten Linien zu zeichnen, muß man die Festlegung des Zeichenmuster ändern. Dafür steht die Funktion SetDrPt() zur Verfügung:

```
SetDrPt(rasp,wert);
```

In diesem Fall wird <wert> als vierstelliger Hex-Wert angegeben, also in dem Format <0xnnnn>. <0xffff> als Zeichenmuster erzeugt "solide" Linien in der Farbe des APen, <0x0000> ebensolche in der Farbe des BPen. Alle möglichen Zwischenkombinationen ergeben andersartig und vielfältig gepunktete Linien.
Nicht ganz so einfach hat man es, wenn man ein Füllmuster festlegen möchte, mit dem die Flächen gefüllt werden sollen. Dieses Füllmuster muß nämlich, wie auch die Farbliste, in einem Array definiert werden. Dies bedeutet für den Programmierer ein bißchen Arbeit, denn man muß sich logisch vergegenwärtigen, welches Resultat eine bestimmte Datenfolge erbringen könnte. Grundsätzlich besteht ein Füllmuster aus einem Array von UWORDs, deren einzelne Bits jeweils einzelne Pixel des Füllmusters beeinflussen. Ein Beispiel hierfür liefert das <fillpattern>-Array in colors&patterns.c. Ein solches Array ist in der Größe auch nicht frei, sondern nur in bestimmten Schritten wählbar. Dies soll anhand des Aufrufs SetAfPt(), welcher das Füllmuster festlegt, erläutert werden:

```
SetAfPt(rasp,&fillpattern[0],exponent)
```

Als zweiten Parameter muß diese Funktion die Adressse des ersten Elements des Füllmuster-Arrays erhalten. Der dritte Parameter stellt einen Wert dar, welcher als Exponent von zwei die Anzahl der UWORDs im Array ergibt. Dabei ist ersichtlich, daß man nur bestimmte Anzahlen von UWORDs festlegen kann, nämlich 2, 4, 8, 16 etc. Die verschiedenen Bits der UWORDs beeinflussen dann einzelne Punkte, wobei ein gesetztes Bit bewirkt, daß der Punkt im APen gezeichnet wird, während ein Ungesetztes im BPen dargestellt wird. Dafür hier noch ein einfaches Beispiel: Enthält Ihr Füllmuster-Array die vier UWORDs <0xff00,0xff00,0x00ff,0x00ff>, so stellen sich diese als Bitkombinationen folgendermaßen dar:

```
=====================================================================
LISTING  VON :                                           Seiten ____
simple_graphics.c            1376 rwed Today      15:59:46
====================================================== 3.4 a ======

/* Standard-Includefile mit Display-Funktionen und       */
/* den noetigen globalen Variablen                       */
#include <screen&window.h>

/* Includefile fuer Aztec-C                               */
/* Bei Verwendung von Lattice-C nicht einbinden          */
#include <functions.h>

struct RastPort* rasp;

SHORT pkt_liste[] =
      { 160,180,
        220,200,
        220,230,
        160,250,
        100,230,
        100,200,
        160,180 };

```

```
UWORD 1:1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
UWORD 2:1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
UWORD 3:0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
UWORD 4:0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
```

Sie erhalten damit ein Füllmuster in den Ausmaßen 16 mal 4 Pixel, das eine Darstellung von flachen versetzten Rechtecken in den Farben des APen und des BPen auf den Bilschirm bringt. Anhand einer solchen Bittabelle läßt sich das gewünschte Muster ohne viele Versuche am einfachsten festlegen. Der Wert für <exponent> wäre in diesem Beispiel 2.

Die Füllmuster müssen allerdings nicht auf zwei Farben beschränkt werden, denn es besteht die Möglichkeit, ein Füllmuster in so vielen Farben zu erzeugen, wie der Screen, auf dem es gezeichnet wird, hat.

Dazu muß man ein zweidimensionales Array festlegen, in dem eine wie oben beschriebene UWORD-Kombination für jede Bitplane des Screens festgelegt werden muß. Als Beispiel hierfür dient das Array <multicolorpattern>im Listing colors&patterns.c. Um ein solches vielfarbiges Muster zu laden, bedient man sich des Befehls SetAfPt() wie oben angegeben, mit dem Unterschied, das man <exponent> mit einem negativen Vorzeichen versehen muß. Der Betrag von <exponent> hängt allerdings weiter von der Anzahl der UWORDs pro Kombination ab. Jede dieser Kombinationen ist dann für die Bits in einer der Bitplanes des Screens zuständig.

Bei manchen Anwendungen kann es für den Programmierer von Interesse sein, nur bestimmte Bitplanes anzusprechen, um dieunterschiedlichsten Effekte zu erzeugen. Für diesen Zweck stellt das Betriebssystem den Befehl SetWrMsk() zur Verfügung, mit dem einzelne Bitplanes des Screens von den Zeichenvorgängen ausgeschlossen werden können. Dieser Aufruf geht folgendermaßen vonstatten:

```
SetWrMsk(rasp,mask);
```

<mask> wird in Form eines zweistelligen Hex-Wertes übergeben, in dem jedes Bit über die Aktivierung einer einzelnen Bitplane entscheidet. Da ein Screen maximal fünf Bitplanes enthalten kann, sind maximal die fünf niedrigsten Bits von Bedeutung, wobei

```
main()
{
 int i,n;

 make_display();

 rasp = grafikwindow->RPort;

 SetAPen(rasp,1L);

/* Positionieren des Grafikcursors                              */
 Move(rasp,25L,170L);

/* Schleife fuer eine Zickzacklinie                             */
 for(i=20;i<261;i+=20)
 {
  SetAPen(rasp,(long)i/20);
  Draw(rasp,25L+(i-10),130L);
  Draw(rasp,25L+i,170L);
 }

/* Schleife fuer verschiedene Ellipsen                          */
 for(i=0;i<131;i+=5)
 {
  SetAPen(rasp,(long)i/5);
  DrawEllipse(rasp,160L,70L,140L-i,25L);
 }

/* Doppelschleife fuer eine Reihe bunter Pixel                  */
 for(n=0;n<8;n++)
  for(i=0;i<32;i++)
  {
   SetAPen(rasp,(long)i);
   WritePixel(rasp,32L+(i+1+(n*32)),110L);
  }

 SetAPen(rasp,6L);

/* Positionieren des Grafikcursors                              */
 Move(rasp,160L,180L);

/* Mehrere Linien in einer Farbe zeichnen                       */
 PolyDraw(rasp,7L,&pkt_liste[0]);

 Wait(1L<<grafikwindow->UserPort->mp_SigBit);

 end_display();

 return(0);
}

                    --- END OF FILE : simple_graphics.c ---
===================================================================

===================================================================
LISTING  VON :                                         Seiten
area_graphics.c               2342 rwed Today       15:59:24
=================================================== 3.4  a ======

/* Standard-Includefile mit Display-Funktionen und              */
/* den noetigen globalen Variablen                              */
#include <screen&window.h>

/* Includefile fuer Aztec-C                                     */
/* Bei Verwendung von Lattice-C nicht einbinden                 */
#include <functions.h>

struct RastPort* rasp;
struct AreaInfo a_info;
struct TmpRas t_ras;
UWORD areabuf[500];
BYTE* area_plane;

main()
{
 int i;

/* Screen und Window in LoRes u. 32 Farben oeffnen              */
 make_display();

/* Zuweisung der RastPort-Struktur                              */
 rasp = grafikwindow->RPort;

/* Initialisierung der AreaInfo-Struktur und Eroeff-            */
/* nung einer Bitplane fuer die Fuellbefehle                    */
 InitArea(&a_info,&areabuf[0],200L);
 area_plane = (BYTE *)AllocRaster(320L,256L);

/* Zuweisung der TmpRas- und AreaInfo-Struktur auf              */
/* den benutzten RastPort                                       */
```

ein gesetztes Bit eine Bitplane aktiviert und ein ungesetztes Bit eine Bitplane abschaltet. Setzt man als <mask> beispielsweise den Wert <0xfb>, welcher dem Binärwert <11111011> entspricht, so bewirkt dies, daß die dritte Bitplane von den Zeichenvorgängen ausgeschlossen wird. Die Auswirkungen eines solchen Aufrufs können Sie in colors&patterns beobachten.

Das Betriebssystem stellt dem Anwender auch verschiedene Zeichenmodi zur Verfügung, mit denen die Verknüpfung bereits gesetzter und darauf gezeichneter Bits bestimmt werden kann. Um den Modus zu wählen, bedient man sich der Funktion SetDrMd():

```
SetDrMd(rasp,modus);
```

Die Werte, die als <modus> übergeben werden können, sind in den Includefiles vordefiniert. Im einzelnen kann der Anwender folgende Modi auswählen:

JAM1:

Dieser Modus bewirkt, daß beim Zeichnen jedes neue Pixel im APen gesetzt wird. Verwenden Sie ein Zeichen- oder Füllmuster, so werden die ungesetzten Bits des Musters ignoriert.

JAM2:

Bei diesem Modus werden der APen und der BPen zum Zeichnen eingesetzt. Für jedes gesetzte Bit im Zeichen- oder Füllmuster wird der APen eingesetzt, für jedes ungesetzte Bit der BPen Dieser Modus ist übrigens voreingestellt.

COMPLEMENT:

COMPLEMENT bewirkt, daß für jedes im Füllmuster gesetzte Bit die bereits in den Bitplanes gesetzten Bits um-gekehrt werden - aus einem Bit 1 wird ein Bit 0 und aus einem Bit 0 ein Bit 1.

INVERSVID:

Bei Anwendung dieses Modus werden die Auswirkungen und die Rolle des APen und des BPen miteinander vertauscht.

Wollen Sie einen der letzten zwei Modi aktivieren, so sollten Sie im allgemeinen beim Aufruf von SetDrMd() den gewünschten JAM-Modus mit angeben. Ein typischer Aufruf sieht dann folgendermaßen aus:

```
 rasp->AreaInfo = &a_info;
 rasp->TmpRas = &t_ras;

/* TmpRas-Struktur initialisieren                              */
 InitTmpRas(&t_ras,area_plane,(long)RASSIZE(320,256));

/* Schleife fuer versch. gefuellte Flaechen                    */
 for(i=0;i<106;i+=5)
 {
  SetAPen(rasp,(long)i/5);
  AreaMove(rasp,80L+(i*2),120L+i);
  AreaDraw(rasp,190L+i,120L+i);
  AreaDraw(rasp,200L+i,200L+(i/2));
  AreaDraw(rasp,90L+(i*2),200L+(i/2));
  AreaEnd(rasp);
 }

/* Punktpuffer muss vor AreaEllipse-Aufruf leer sein */
 for(i=0;i<500;i+=1)
  areabuf[i] = (UWORD)0;

/* Schleife fuer versch. gefuellte Ellipsen                    */
 for(i=0;i<131;i+=5)
 {
  SetAPen(rasp,(long)i/5);
  AreaEllipse(rasp,160L,70L,140L-i,25L);
  AreaEnd(rasp);
 }

/* Wieder den Punktpuffer leeren...                            */
 for(i=0;i<500;i+=1)
  areabuf[i] = (UWORD)0;

/* ...und noch ein paar Ellipsen (nur ganz kleine)     */
 for(i=0;i<25;i+=2)
 {
  SetAPen(rasp,(long)i/2);
  AreaEllipse(rasp,160L,70L,14L,25L-i);
  AreaEnd(rasp);
 }

/* Schleife fuer versch. gefuellte Rechtecke                   */
 for(i=0;i<32;i++)
 {
  SetAPen(rasp,(long)i);
  RectFill(rasp,10L+i,120L+(i*2),20L+(i*2),130L+(i*3));
 }

/* Nun den Hintergrund fuellen                                 */
 SetAPen(rasp,5L);
 Flood(rasp,1L,15L,15L);

/* Intuition wartet auf Betaetigung des Close-Gadgets*/
 Wait(1L<<grafikwindow->UserPort->mp_SigBit);

/* Speicherplatz ans System zurueckgeben                       */
 FreeRaster(area_plane,320L,256L);
 end_display();

 return(0);
}

                          --- END OF FILE : area graphics.c ---
=================================================================

=================================================================
LISTING VON :                                       Seiten
colors&patterns.c          4494 rwed Today      15:58:49
==================================================== 3.4 a ======

/* Standard-Includefile mit Display-Funktionen und   */
/* den noetigen globalen Variablen                   */
#include <screen&window.h>

/* Includefile fuer (u.a.) die Definition von SetOPen*/
#include <graphics/gfxmacros.h>

/* Includefile fuer Aztec-C                          */
/* Bei Verwendung von Lattice-C nicht einbinden      */
#include <functions.h>

struct ViewPort* viep;
struct RastPort* rasp;
struct AreaInfo a_info;
struct TmpRas t_ras;
UWORD areabuf[500];
BYTE* area_plane;
UWORD farbliste[] = { 0x000,0xf00,0x0f0,0x00f,
                      0xfff,0xfb0,0x0db,0xc1f,
```

```
                             0xd00,0xf80,0xfd0,0xff0,
                             0xf90,0xbf0,0x8e0,0x2c0,
                             0x0b1,0x0bb,0x1fb,0x6fe,
                             0x6ce,0x61f,0x06d,0x91f,
                             0xf1f,0xfac,0xdb9,0xc80,
                             0xa87,0xccc,0x999,0xab6 };

UWORD fillpattern[] = { 0xff00,0xff00,0x00ff,0x00ff,
                               0xf0f0,0xf0f0,0x0f0f,0x0f0f };

UWORD multicolorpattern[5][8] = {
                           { 0x0000,0x0000,0xffff,0xffff,
                             0x0000,0x0000,0xffff,0xffff },
                           { 0x0000,0x0000,0x0000,0x0000,
                             0xffff,0xffff,0xffff,0xffff },
                           { 0xff00,0xff00,0xff00,0xff00,
                             0xff00,0xff00,0xff00,0xff00 },
                           { 0x0000,0x0000,0x0000,0x0000,
                             0x0000,0x0000,0x0000,0x0000 },
                           { 0x0000,0x0000,0x0000,0x0000,
                             0x0000,0x0000,0x0000,0x0000 } };


main()
{
 int i;

 make_display();

 rasp = grafikwindow->RPort;

/* ViewPort-Zuweisung                                          */
 viep = &grafikscreen->ViewPort;

 InitArea(&a_info,&areabuf[0],200L);
 area_plane = (BYTE *)AllocRaster(320L,256L);

 rasp->AreaInfo = &a_info;
 rasp->TmpRas = &t_ras;

 InitTmpRas(&t_ras,area_plane,(long)RASSIZE(320,256));

/* Farbpalette in den ViewPort des Screens laden              */
 LoadRGB4(viep,&farbliste[0],32L);

/* 32 Rechtecke in verschiedenen Farben                         */
 for(i=0;i<289;i+=9)
 {
  SetAPen(rasp,(long)i);
  RectFill(rasp,15L+i,20L,25L+i,40L);
 }

 Delay(50L);

/* Farben Nr.1 und Nr.4 sollen getauscht werden                */
 SetRGB4(viep,1L,15L,15L,15L);
 SetRGB4(viep,4L,15L,0L,0L);

 Delay(50L);

/* Rechteck mit Rand zeichnen                                    */
 SetAPen(rasp,4L);
 SetOPen(rasp,2L);
 RectFill(rasp,50L,50L,150L,100L);

 Delay(50L);

/* Nicht eingerahmte...                                          */
 SetAPen(rasp,4L);
 Move(rasp,170L,60L);
 Text(rasp,"Nicht eingerahmt",16L);

/* ...und eingerahmte Texte                                      */
 SetAPen(rasp,3L);
 SetBPen(rasp,2L);
 Move(rasp,170L,85L);
 Text(rasp,"Eingerahmt",10L);

 Delay(50L);
/* Nun werden ein paar gepunktete Linien gezeichnet            */
 SetBPen(rasp,0L);

 SetAPen(rasp,4L);
/* Hierfuer muss das DrawPattern veraendert werden             */
 SetDrPt(rasp,0xcccc);
 Move(rasp,50L,110L);
 Draw(rasp,270L,110L);

 SetAPen(rasp,4L);
```

```
SetDrMd(rasp,JAM2|
COMPLEMENT);
```

Die verschiedenen Darstellungsvariationen, die sich aus den Kombinationen der verschiedenen Modi ergeben, sind so vielfältig, daß man sie schwerlich beschreiben kann. In colors&patterns.c finden Sie ein Beispiel für den COMPLEMENT-Modus, aber um alle Möglichkeiten kennenzulernen, experimetieren Sie am besten selbst ein bißchen herum.

Ein Befehl, der ohne Beispiel in colors&patterns.c ist, ist die Funktion SetRast(). Sie dient dazu, einen RastPort komplett in einer Farbe einzufärben. Aufgerufen wird die Funktion folgendermaßen:

```
SetRast(rasp,index);
```

<index> gibt bei diesem Aufruf die gewünschte Farbe an. Bei einem normalen Window bewirkt SetRast() allerdings, daß das gesamte Window einschließlich der Titelzeile gefärbt wird. Dies läßt sich nur mittels eines GimmeZeroZero-Windows vermeiden; für mehr Information über diesen Window-Typ lesen Sie bitte im Artikel zum Includefile screen&window.h. Damit beim SetRast()-Aufruf auch alles gleichmäßig eingefärbt wird, müssen über SetWrMsk() übrigens alle Bitplanes zum Zeichnen aktiviert sein. So, damit wären die wichtigsten Zeichenfunktionen des Betriebssystems abgehandelt. Mit den vorliegenden Informationen und anhand der Beispielsprogramme, die sich in den abgedruckten Versionen übrigens sowohl mit dem Aztec 3.4a-Compiler als auch mit dem Lattice 3.10-Compiler compilieren lassen, werden Sie sicher in der Lage sein, diese "gewinnbringend" anzuwenden. Natürlich könnte man noch eine ganze Menge mehr über die Strukturen, die der Bildschirm- und Grafikverwaltung zugrundeliegen, sagen, aber der Hobbyprogrammierer, der nicht unbedingt in die Tiefen des Betriebssystems einsteigen möchte, findet in den oben beschriebenen Strukturen und Funktionen genug Informationen, um die Amigagrafik voll auszunutzen, ohne eine Unzahl von Abhandlungen über das Betriebs-

```
 SetDrPt(rasp,0x4444);
 Move(rasp,50L,120L);
 Draw(rasp,270L,120L);

/* Und dasselbe noch einmal, nur mit anderem BPen        */
 SetBPen(rasp,2L);

 SetAPen(rasp,4L);
 SetDrPt(rasp,0xcccc);
 Move(rasp,50L,130L);
 Draw(rasp,270L,130L);

 SetAPen(rasp,4L);
 SetDrPt(rasp,0x4444);
 Move(rasp,50L,140L);
 Draw(rasp,270L,140L);

 Delay(50L);

/* Das Fuellmuster wird geladen                          */
 SetAfPt(rasp,&fillpattern[0],3L);

/* Eine Flaeche wird im Fuellmuster gezeichnet           */
 RectFill(rasp,50L,150L,120L,220L);

 Delay(50L);

/* Das Vielfarbenmuster wird geladen                     */
 SetAfPt(rasp,&multicolorpattern[0][0],-3L);

/* Eine Flaeche wird im Fuellmuster gezeichnet           */
 RectFill(rasp,130L,150L,200L,220L);

 Delay(50L);

/* Nun wird Bitmap 2 desaktiviert...                     */
 SetWrMsk(rasp,0xfb);

/* ...und die letzte Flaeche noch einmal gezeichnet      */
 RectFill(rasp,210L,150L,280L,220L);

 Delay(50L);

/* Alle Bitplanes aktivieren                             */
 SetWrMsk(rasp,0xff);

/* Nun folgen die letzten drei RectFills mit             */
/* veraendertem Zeichenmodus                             */
 SetDrMd(rasp,JAM1|COMPLEMENT);

 SetAfPt(rasp,&fillpattern[0],3L);
 RectFill(rasp,50L,150L,120L,220L);
 Delay(50L);
 SetAfPt(rasp,&multicolorpattern[0][0],-3L);
 RectFill(rasp,130L,150L,200L,220L);
 Delay(50L);
 SetWrMsk(rasp,0xfb);
 RectFill(rasp,210L,150L,280L,220L);

 Wait(1L<<grafikwindow->UserPort->mp_SigBit);

 FreeRaster(area_plane,320L,256L);

 end_display();

 return(0);
}

                    --- END OF FILE : colors&patterns.c ---
==================================================================
```

system wälzen zu müssen. So bleibt dann nur noch zu sagen:
Viel Erfolg bei der Erforschung der Möglichkeiten des Amiga, und viel Spaß mit Ihren selbstgeschriebenen Grafikprogrammen!

VON ANDREAS DIEBOLD

GUT AUSGEDRÜCKT

Der Amiga hat keinen eigentlichen Textmodus. Die Darstellung von Buchstaben und anderen Zeichen erfolgt ausschließlich im Grafikmodus. Der Zeichensatz liegt als Bitmuster im Speicher vor. Aus diesem Bitmuster werden die gewünschten Zeichen an eine bestimmte Stelle im Bildschirmspeicher kopiert. All dies übernimmt der Amiga, ohne Zutun, selbst. Gnädig, wie das Betriebssystem nun einmal ist, stellt es außerdem noch eine Reihe von Funktionen und Strukturen zur Verfügung, mit denen man die Zeichensätze, sprich Fonts, höchst individuell gestalten kann.

Das bedeutet für den Programmierer, daß er nicht nur eine einzige Art von Buchstaben und Zeichen in seinem Programm benutzen kann, sondern auch andere, die schon definiert auf der Workbench bereitliegen. Außerdem ist es möglich, vollkommen neue Zeichensätze zu erstellen und für das System nutzbar zu machen.

Fonts können in folgenden Formen vorliegen: Ein Font, nämlich "topaz.font" befindet sich im ROM, bzw. WORM. Wir alle kennen ihn, denn er ist die Schriftart des CLI und der Workbench sowie aller darauf laufenden Standardapplikationen. Weiterhin befinden sich im Verzeichnis "fonts" einer jeden Workbenchdiskette einige Fonts. Diese nennt man Diskfonts. Diskfonts sind nicht direkt verfügbar, sondern müssen erst von einem Programm in das System eingebunden werden, und liegen dann als Systemfonts vor. Diese sind dann im RAM vorhanden und von allen Tasks nutzbar.

Bei der Benutzung steht ein Font direkt mit einem RastPort in Verbindung. Da jedes Window einen eigenen RastPort besitzt, ist es möglich, auf jedem Window einen anderen Font zu benutzen.

Wichtig für den Gebrauch von Diskfonts ist ein logisches Gerät "FONTS:", auf welchem einige Zeichensätze vorhanden sein sollten. Ein logisches Gerät wird die Directory "fonts" mit dem Befehl ASSIGN. Beim System-start mit der Workbench geschieht dies von selbst. Wer seine Compilerdis-ketten um die "fonts"-Directory er-leichtert hat, um Platz zu gewinnen, muß nun sehen, wie er diese wieder in Reichweite bekommt. Doch der Um-gang mit Fonts lohnt sich, da man sei-nen Programmen eine persönliche Note geben kann, um sie so von der breiten eintönigen Masse abzuheben.

Um Fonts aus C heraus benutzen zu können, benötigt man zwei Headerfiles, "graphics/text.h" und "libraries/diskfont.h", welche die nötigen Struktur- und Flagdefinitionen bereitstellen. Diese Headerfiles hier komplett aufzuführen, ist nicht nötig, da Sie diese in der Include-Directory Ihrer Compilerdisketten finden; allerdings soll ihr Inhalt im Zusammenhang mit den Funktionen näher erläutert werden.

Vorab die Erklärung einer wichtigen Struktur:

```
struct TextAttr aus "graphics/text.h"

Ein Zeiger auf den Fontnamen:
STRPTR TextAttr.ta_Name
Die Zeichenhöhe in Pixeln   :
UWORD TextAttr.ta_YSize
Der Style des Fonts         :
UBYTE TextAttr.ta_Style
Die Flags des Fonts         :
UBYTE TextAttr.ta_Flags
```

Der Style gibt an, in welcher Art die Zeichen dargestellt werden. Auch wenn das Aussehen des Fonts nur in einer Form definiert ist, berechnet das Betriebssystem ein anderes Aussehen. Die Stylevariable selbst ist eine Bitmaske, in welcher die vier niederwertigsten Bits ausschlaggebend sind. Jedes dieser Bits steht, wenn es gesetzt ist, für ein gewisses Merkmal des Fonts. Dafür mögliche Formen sind:

FS_NORMAL
(keinerlei Bits gesetzt
==> Style = 0)
Ein Standardfont, welcher in all anderen Arten umgewandelt werden kann.
FSF_EXTENDED
(drittes Bit gesetzt
==> Style = 8)
Der Font ist in die Breite gezogen.
FSF_ITALIC
(zweites Bit gesetzt
==> Style = 4)
Kursive Darstellung.
FSF_UNDERLINED
(erstes Bit gesetzt
==> Style = 2)
Unterstrichen.
FSF_BOLD
(nulltes Bit gesetzt
==> Style = 1)
Fettdruck.

Diese Styles sind beliebig kombinierbar, wobei EXTENDED allerdings nicht immer möglich ist.
Die Flags sagen etwas über Beschaffenheit des Fonts aus, etwa, ob es sich um einen Diskfont oder einen Romfont handelt.
Diese Struktur dient unter anderem zum Auffinden eines Fonts. Man legt in TextAttr fest, welcher Art er sein soll. Dabei wird der, in Bezug auf <ta_YSize>, am nächsten passende Font gewählt. Sind die gewünschten Styles nicht verfügbar, so erzeugt das System diese. Dabei sind allerdings keine Stylemerkmale rückgängig zu machen, wenn eins oder mehrere in der Fontdefinition vorgesehen waren.

Funktionen zum Gebrauch von Fonts

(Allgemeine Anmerkung: In der Beschreibung der einzelnen Funktionen finden Sie hinter <Ü:> die Übergabeparameter. Die Art des Parameters ist im C-Syntax dargestellt. Hinter <R:> befindet sich eine Beschreibung dessen, was aus der Funktion resultiert. Die Funktionen erwarten die Parameter in bestimmten Registern. Der C-Programmierer braucht sich darum nicht zu kümmern. Für Assembler-Pro-

```
==================================================================
LISTING  VON :                                         Seiten ____
makediskfontfile.c            2194 rwed Today      16:56:16
===================================================== 3.4 a ======

/* Dieses Programm erzeugt den Header fuer einen
 * DiskFont. Man findet solche Files als xxx.font
 * innerhalb des FONTS: Verzeichnisses.
 * Dieser File besteht aus:
 *   1. einem  WORD  ein Identifier, der Hex 0F00 ist.
 *   2. einem  WORD  in dem die Anzahl der diversen
 *                   Hoehen zu dem Font steht.
 *
 * dann f/r jede Hoehe seperat:
 *   1. ein 256 Byte gro_es Feld, in dem der Pathname
 *                   zum eigentlichen Font steht.
 *   2. ein WORD     die Pixelhoehe.
 *   3. ein UBYTE    die Flags.
 *   4. ein UBYTE    das Hex 60 betragen sollte.
 *
 *****************************************************/

#include <exec/types.h>

/* nur fuer Aztec C */
#include <functions.h>

#include <libraries/dos.h>

struct FileHandle *fh;

/* Identifier fuer den Header */
UBYTE id[] = { 0x0f, 0x00 };

UBYTE pathname[256], flags, name[80];
UWORD fontheight,number,buf;
UBYTE fontkind = 0x60; /* Das bedeutet einen Diskfont,
                          der nicht resident ist */

main()
{
 SHORT i;

 printf("\n\nWie soll der Fontname sein ?:");
 printf(">>.font nicht vergessen<<\n");
 scanf("%s",&name[0]);

 fh = (struct FileHandle *)Open(&name[0],MODE_NEWFILE);

 Write(fh,&id[0],2L);

 printf("\n\nWieviel definierte Groessen gibt es ?:");
 scanf("%d",&number);

 Write(fh,(UBYTE *)&number,2L);


 for(i=1;i<=number;i++)
  {
   printf("\nEingaben f/r die %d. Groesse\n\n",i);
   printf("\n\nBitte den Pathname innerhalb der ");
   printf("Fontsdirectory :\n");
   scanf("%s",&pathname[0]);

   Write(fh,&pathname[0],256L);

   printf("\n\nHoehe des %d.Fonts ?:",i);
   scanf("%d",&fontheight);

   Write(fh,&fontheight,2L);

/****************************************************
 * Die Flags als Ascii-Wert setzen hei_t, wenn Sie *
 * z.B. das fuenfte und das zweite Bit setzen      *
 * wollen, ist der Ascii-Wert dieser Bitmaske      *
 * 2^5 + 2^2 = 32 + 4 = 36                         */

   printf("\n\nFlags als ASCII-Wert :");
   scanf("%d",&buf);
   flags = (UBYTE)buf;

   Write(fh,&flags,1L);

   Write(fh,&fontkind,1L);
  }
```

```
  printf("\n\nDer File ist geschrieben, und liegt");
  printf("\nim aktuellen Verzeichniss\n\n");
  Close(fh);
  return(NULL);
}


                        --- END OF FILE : makediskfontfile.c ---
=====================================================================
=====================================================================
LISTING  VON :                                         Seiten ____
makediskfont.asm                2760 rwed Today     16:58:06
=====================================================================

; Dient zum Erzeugen von Diskfonts.
; File mu_ assembliert und gelinkt werden
;***********************************************
;INCLUDE "exec/types.i"
;INCLUDE "exec/nodes.i"
;INCLUDE "libraries/diskfont.i"

DFH_ID EQU $0f80
MAXFONTNAME EQU 32
FPF_PROPORTIONAL EQU 32
FPF_DESIGNED EQU 64
NT_FONT EQU 12
                            ;*************************
 MOVEQ   #0,D0              ;* ABBRUCH BEIM
 RTS                        ;* STARTVERSUCH
                            ;*************************

 DC.L    0                  ;*************************
 DC.L    0                  ;*
 DC.B    NT_FONT            ;*  HIER
 DC.B    0                  ;*  WERDEN
 DC.L    fontName           ;*  STRUKTURN
 DC.W    DFH_ID             ;*  INITIALISIERT,
 DC.W    1                  ;*  DIE DEM
 DC.L    0                  ;*  SYSTEM DIENEN.
                            ;*
fontName:                   ;*
 DS.B    MAXFONTNAME        ;*
                            ;*
font:                       ;*  <-- Beginn der
 DC.L    0                  ;*    TextFont Struktur
 DC.L    0                  ;*
 DC.B    NT_FONT            ;*  KEINE AENDERUNGEN
 DC.B    0                  ;*
 DC.L    fontName           ;*  SINNVOLL
 DC.L    0                  ;*
 DC.W    fontEnd-font       ;**********************

;***************************************************
;
;  AB HIER WIRD DER NEUE FONT BESTIMMT
;
;***************************
                              ;*
 DC.W    8                    ;* Zeichenhoehe in Pixel
 DC.B    0                    ;* Bestimmung des Styles
 DC.B    FPF_DESIGNED+FPF_PROPORTIONAL ;*  und der Flags
 DC.W    14                   ;* Normalbreite in Pixel
 DC.W    6                    ;* Lage der Baseline
                              ;*
 DC.W    1                    ;*
 DC.W    0                    ;* Zaehler mu_ 0 sein
 DC.B    97                   ;* ASCII-Wert f/r 1.Zeichen
 DC.B    100                  ;* und fuer das letzte
 DC.L    fontData             ;* nicht aendern
 DC.W    8                    ;* Modulo Wert
                              ;*
 DC.L    fontLoc              ;* nicht aendern
                              ;*
 DC.L    fontSpace            ;* nicht aendern
 DC.L    fontKern             ;* nicht aendern

fontData:    ;* Das eigentliche Aussehen des Fonts

 DC.W    $071C0,$08040,$070FF,$0F000
 DC.W    $0FBE3,$0E0E0,$0F8C0,$03000
 DC.W    $07FCF,$0F9F3,$026C0,$03000
 DC.W    $03F9F,$0FFFF,$0FFC0,$03000
 DC.W    $01F0E,$0B9F3,$026C0,$03000
 DC.W    $00E00,$080E0,$020C0,$03000
```

grammierer sind die entsprechenden Register hinter den Variablenbeschreibungen in Klammern vermerkt.

Ein anderes Problem, das sich nur in Assembler stellt, ist die Notwendigkeit, die einzelnen Registerinhalte zu retten, bevor die Funktionsparameter hineingeschrieben werden. Um die folgenden Funktionen in Assembler aufzurufen, muß sich ein Zeiger auf GfxBase in A6 befinden.

Manche Funktionen liefern eine Variable <fehler> zurück. Ist diese ungleich Null, so ist die Ausführung der Funktion mißlungen.

Es sei auch hier noch einmal erwähnt, daß die Systemfunktionen nahezu grundsätzlich Zeiger oder LONG-Variablen (= 32 Bit) verlangen und zurückgeben.)

AddFont (textfont);

Ü: struct TextFont *textfont (A1).
R: Ein Font wird für das System verfügbar gemacht.

An AddFont() wird eine Struktur TextFont übergeben. Diese Struktur muß in einem Speicherbereich liegen, der allen Tasks zugänglich ist. Einen solchen reserviert man mit der Funktion AllocMem(), in die das Flag <MEMF_PUBLIC> eingebracht werden muß (siehe Listing addmyfont.c). TextFont stellt die komplette Definition eines Fonts dar, der mit AddFont() verfügbar gemacht wird. AddFont() überprüft leider nicht, ob diese Definition gültig ist.

AskFont (rastport, textattr);

Ü: struct RastPort *rastport (A1).
struct TextAttr *textattr (A0).
R: <textattr> wird mit den Fontdaten gefüllt, die mit <rastport> in Verbindung stehen.

In der Struktur TextAttr befinden sich die wichtigsten Informationen zur Identifizierung eines Fonts. Mit AskFont() wird eine TextAttr-Struktur mit den zu einem angegebenen RastPort gehörenden Fontdaten gefüllt. So kann man prüfen, welcher Font in welcher Art auf einem Window aktiv ist.

enable = AskSoftStyle (RastPort);

Ü: struct RastPort *rastport (A1).
R: UBYTE enable (D0).

Hier erhalten wir eine Bitmaske. Diese gibt an, welche Stylebits geändert werden dürfen und welche nicht. Bei einem Font, der als normal <FS_NORMAL> konstruiert wurde, kann alles geändert werden. <enable> stellt das Inversum des im Font festgelegten Style dar und wird zur nachträglichen Änderung durch SetSoftStyle() gebraucht.

fehler= AvailFonts(buffer, buffergröße, fonttyp);

Ü: struct AvailFontsHeader *buffer (A0).
LONG buffergröße (D0).
UWORD fonttyp (AFF_DISK und/oder AFF_MEMORY) (D1).
R: LONG fehler (D0).

Mit dieser Funktion kann man prüfen, welche Fonts verfügbar sind, wobei allerdings nicht garantiert ist, ob sie wirklich benutzbar sind. Der Programmierer muß der Funktion einen Speicherbereich zur Verfügung stellen, welcher die Rückgabewerte der Funktion aufnimmt. Das Bereitstellen eines Speicherbereichs sollte mit AllocMem() geschehen. Jeder verfügbare Font belegt in <buffer> ca.30 Byte, daher sollte eine Größe von 1000 Bytes vollkommen ausreichend sein. Mit <fonttyp> bestimmt man, wo nach verfügbaren Fonts gesucht wird. Eine Kombination der Suche im Speicher und auf Diskette ist möglich. In <buffer> befinden sich, nach erfolgreicher Ausführung von AvailFonts(), einige Strukturen. Die erste ist vom Typ AvailFontsHeader. Diese Struktur besitzt nur ein Element, nämlich das UWORD<afh_NumEntries>, welches Auskunft darüber gibt, wieviele Strukturen des Typs AvailFonts nun in <buffer> folgen. Für jeden gefundenen Font

```
 DC.W     $00403,$0E040,$0F8FF,$0F000
 DC.W     $00000,$00000,$00000,$00000
 DC.W     $00000,$00000,$00000,$00000

fontLoc:            ;* Offset und Breite der Zeichen

 DC.L     $00000000B,$0000B000B,$000160007,$0001D000B
 DC.L     $00028000C

fontSpace:          ;* Breite einzelner Zeichen

 DC.W     000012,000012,000008,000012,000013

fontKern:       ;* Beginn des Zeichens in der Matrix
 DC.W     000001,000001,000001,000001,000001

fontEnd:        ;* Selbstredend
 END


                     --- END OF FILE : makediskfont.asm ---
:===================================================================
====================================================================
LISTING  VON :                                           Seiten ____
addmyfont.c                   4596 rwed Today      18:08:40
====================================================================

/****************************************************************
 *                                                              *
 * Dieses Programm initialisiert einen Font und linkt *
 * ihn direkt in das System, unter Zuhilfenahme                 *
 * von AddFont(). Als Zeichen seiner Existenz ist               *
 * eine Window-Titlebar mit dem Fontnamen auf dem               *
 * WorkbenchScreen zu sehen. Solange das Close-Gadget *
 * nicht betaetigt wird, ist dieser Font fuer alle              *
 * Tasks benutzbar. Nach dessen Betaetigung wird der            *
 * Font entfernt und aller Speicher, der benutzt                *
 * wurde, wieder freigegeben.                                   *
 * Das Programmm muss mit 'RUN ADDMYFONT' gestartet             *
 * werden, damit das CLI weiter zur Eingabe zur                 *
 * Verfuegung steht.                                            *
 ****************************************************************/


struct IntuitionBase    *IntuitionBase;
struct GfxBase          *GfxBase;
struct TextFont         *tf;
struct Window           *cleanup_w;
struct NewWindow nw =
 { 0,20,240,10,-1,-1,CLOSEWINDOW,WINDOWDRAG|WINDOWDEPTH
   |WINDOWCLOSE,NULL,NULL,NULL,NULL,NULL,0,0,0,0,
   WBENCHSCREEN };

/* Das Aussehen des Fonts */
UWORD cdat[] = { 0xe00f, 0x803f, 0xffe4, 0x63e0,
                 0x3019, 0x803c, 0x01b4, 0x9080,
                 0x1821, 0x8030, 0x032c, 0x9080,
                 0x0c60, 0x0c60, 0x0624, 0x6080,
                 0x06c0, 0x3060, 0x0c00, 0x0000,
                 0x0380, 0x1fc0, 0xff12, 0x93c0,
                 0x06c0, 0x00c0, 0x300c, 0x7080,
                 0x0c60, 0x0180, 0xe00c, 0x1100,
                 0x1830, 0xc301, 0x8012, 0x63c0,
                 0x3018, 0xc603, 0x00c0, 0x0000,
                 0xe00e, 0x7c0f, 0xffc0, 0x0000,
                 0x0000, 0x0000, 0x0000, 0x0000,
                 0x0000, 0x0000, 0x0000, 0x0000,
                 0x0000, 0x0000, 0x0000, 0x0000,
                 0x0000, 0x0000, 0x0000, 0x0000
              };

/* Lage der einzelnen Zeichen */
UWORD cloc[] = { 0x0000, 0x000f, 0x000f, 0x000d,
                 0x001c, 0x000e, 0x002a, 0x0011 };

/* Breite der einzelnen Zeichen */
UWORD cspc[] = { 17,15,16,19 };

UWORD cker[] = { 2,2,2,2 };

char  fontname[32];

main()
 {
  LONG err;
```

```
 UWORD i,*ptr;

 IntuitionBase = (struct IntuitionBase * )
               OpenLibrary("intuition.library",0L);
 if(!IntuitionBase)
ende();

 GfxBase = (struct GfxBase * )
               OpenLibrary("graphics.library",0L);
 if(!GfxBase)
ende();

/* Die TextFont Struktur wird in einen Speicherblock
   gelegt, der allen Tasks zugaenglich ist (MEMF_PUBLIC)*/
 tf = (struct TextFont *)AllocMem(
               (LONG)( sizeof(struct TextFont) )
                                   ,MEMF_PUBLIC);

 if(!tf){printf("\nMEM - FAULT\n");ende();}

 str_copy("my.font",fontname);/* Der Name des Fonts */

 /* Nun folgt die Initialisierung der TextFont-Struktur.
    Die ersten 7 Elemente koennen auch bei Ihren Fonts
    so belassen werden. */
 tf->tf_Message.mn_Node.ln_Succ = NULL;
 tf->tf_Message.mn_Node.ln_Pred = NULL;
 tf->tf_Message.mn_Node.ln_Type = NT_FONT;
 tf->tf_Message.mn_Node.ln_Pri  = NULL;
 tf->tf_Message.mn_Node.ln_Name = fontname;
 tf->tf_Message.mn_ReplyPort    = NULL;
 tf->tf_Message.mn_Length       = (UWORD)
                       ( sizeof(struct TextFont) +
                         sizeof(cdat) +
                         sizeof(cloc) +
                         sizeof(cspc) +
                         sizeof(cker) );
 /* Ab hier den eigenen Font definieren */
 tf->tf_YSize      = 14;
 tf->tf_Style      = NULL;
 tf->tf_Flags      = FPF_PROPORTIONAL|FPF_DESIGNED;
 tf->tf_XSize      = 19;
 tf->tf_Baseline   = 12;
 tf->tf_BoldSmear  = 1;
 tf->tf_Accessors  = NULL;       /* bleibt gleich */
 tf->tf_LoChar     = 88;
 tf->tf_HiChar     = 90;
 tf->tf_CharData   = (APTR)cdat; /* bleibt gleich */
 tf->tf_Modulo     = 8;
 tf->tf_CharLoc    = (APTR)cloc; /* bleibt gleich */
 tf->tf_CharSpace  = (APTR)cspc; /* bleibt gleich */
 tf->tf_CharKern   = (APTR)cker; /* bleibt gleich */

 AddFont(tf);

 nw.Title = (UBYTE *)fontname;

 cleanup_w = (struct Window *)OpenWindow(&nw);
 if(!cleanup_w)ende();

 /* Wait() bewirkt dass der Task im Pause-Zustand
    ist und nicht bremsend auf Anderes wirkt. */
 Wait(1L<<cleanup_w->UserPort->mp_SigBit);

 RemFont(tf);

 ende();
}

ende()
{
 if(cleanup_w)       CloseWindow(cleanup_w);
 if(GfxBase)         CloseLibrary(GfxBase);
 if(IntuitionBase)    CloseLibrary(IntuitionBase);
 if(tf)
       FreeMem(tf,(long)sizeof(struct TextFont));
 exit(1L);
}

str_copy(a,b)
char *a,*b;
{
while(*b++ = *a++);
}                   --- END OF FILE : addmyfont.c ---
=================================================================
```

gibt es einen Eintrag, also auch für verschiedene Größen und Arten jeweils einen separaten. Sucht man sowohl auf Diskette, als auch im Speicher, werden Doppelnennungen möglich. Die Struktur AvailFonts bietet in ihrem Element <af_Type> darüber Aufschluß, wo der in ihr erwähnte Font gefunden wurde. <af_Type> nimmt dann also entweder den Wert AFF_DISK oder AFF_MEMORY an. Das zweite Element von AvailFonts, <af_Attr>, ist ein Zeiger auf die schon besprochene Struktur TextAttr, welche eine genaue Beschreibung des Fonts liefert. Nach der Auswertung dieser Informationen sollte man nicht vergessen, den für <buffer> reservierten Speicherplatz wieder freizugeben.

ClearEOL(rastport);

Ü: struct RastPort *rastport (A1).
R: Eine Zeile wird gelöscht.

Diese Funktion arbeitet mit dem Grafikbefehl Move() zusammen. Nachdem der Grafikcursor mit Move() positioniert wurde, löscht diese Funktion von dort aus bis zum Ende der Zeile (E(nd)O(f)L(ine)). Die Zeilenhöhe, in der gelöscht wird, entspricht der Höhe des Fonts, der auf dem genannten RastPort aktiv ist.

ClearScreen(rastport);

Ü: struct RastPort *rastport (A1).
R: Der gesamte Darstellungsbereich des RastPorts wird gelöscht.

Ab dem durch Move() gesetzten Grafikcursor wird bis zum rechten und zum unteren Bildrand gelöscht. Löschen heißt, daß alle Pixel mit der Farbe 0 gesetzt werden. Ist der genannte RastPort allerdings im JAM2-Modus, werden die Pixel in der Farbe gesetzt, die im BPen, also mit der Funktion SetBPen(), definiert wurde.

CloseFont(textfont);

Ü: struct TextFont *textfont (A1).
R: Der aufrufende Task beendet seinen Zugriff auf den Font.

Ein geöffneter TextFont wird für den aufrufenden Task geschlossen. Dies sollte immer getan werden, um die Systemressourcen nicht überzustrapazieren. Systemfonts werden aber nicht aus dem Speicher entfernt. Sie bleiben also für andere Tasks erhalten.

textfont = OpenDiskFont (textattr);

Ü: struct TextAttr *textattr (A1).
R: struct TextFont *textfont (D0).

Zuerst muß eine Struktur TextAttr mit den gewünschten Werten gefüllt werden. Diese Struktur, bzw. deren Adresse, wird dann übergeben. Das System sucht nun einen Font, der den in TextAttr angegebenen Parametern entspricht. Zuerst wird im Speicher gesucht, dann in “FONTS:”. Wird die in TextAttr angegebene Höhe <YSize> nicht gefunden, so wird ein Font der nächsten Höhe genommen. War der Font nicht im Speicher vorhanden, aber auf Diskette, so ist er von nun an als Systemfont verfügbar. <textfont> ist ein Zeiger auf seine Position im Speicher, welcher für andere Funktionen benötigt wird. Ist dem in TextAttr definierten Wunsch in keiner Weise genüge zu tun, so ist <textfont> nach der Ausführung Null, d.h., ein derartiger Font ist in keiner Weise verfügbar.

TextFont= OpenFont(TexAttr);

Wie OpenDiskFont(), doch wird nicht auf der Diskette gesucht, falls der spezifizierte Font nicht im Speicher liegt.

fehler = RemFont(textfont);

Ü: struct textFont *textfont (A1).
R: LONG fehler (D0).

Der angegebene Font, welcher als Systemfont vorhanden sein sollte, wird aus dem Speicher entfernt und aus der Systemfontliste gestrichen.
Ein mit AddFont(), OpenDiskFont() oder OpenFont() ins System eingebun-

```
=======================================================================
LISTING  VON :                                              Seiten ____
showfont.c                        4720 rwed Today      16:42:53
==================================================== 3.4 a ======

#include <intuition/intuitionbase.h>
#include <libraries/diskfont.h>

/* nur fuer Aztec C */
#include <functions.h>

#include <libraries/dos.h>

#define RP font_window->RPort

struct DosBase            *DosBase;
struct DiskfontBase       *DiskfontBase;
struct IntuitionBase      *IntuitionBase;
struct GfxBase            *GfxBase;

struct TextFont *tf;
struct TextAttr textattr;

/* Ein einfaches Workbench Window auf dem die Fonts
   gezeigt werden */
struct NewWindow windowdata =
 {
  20,80,600,80,0,1,CLOSEWINDOW,WINDOWDRAG|WINDOWCLOSE|
  SIMPLE_REFRESH,NULL,NULL,(UBYTE *)"FONT CHECK",NULL,
  NULL,600,80,600,80,WBENCHSCREEN
 };

struct Window *font_window;

LONG str_len(),dm;


main()
 {
  SHORT buf;
  UBYTE *fsp,fstr[30];
  UWORD fheight;

  if( (DiskfontBase = (struct DiskfontBase *)
         OpenLibrary("diskfont.library",0L)) == NULL )
                  ende();

  if( (IntuitionBase = (struct IntuitionBase *)
         OpenLibrary("intuition.library",0L)) == NULL )
                  ende();

  if( (GfxBase = (struct GfxBase *)
         OpenLibrary("graphics.library",0L)) == NULL )
                  ende();

  fsp = (UBYTE *)&fstr[0];

/*****************************************************
 * Einige Abfragen und initialisieren der TextAttr  *
 * Struktur.                                         */

  printf("\n\n Fontname ? :");
  scanf("%s",fsp);

  textattr.ta_Name = fsp;

  printf("\n\n Fonthvhe ? :");
  scanf("%d",&buf);
  fheight = (UWORD)buf;

  textattr.ta_YSize = fheight;

  printf("\n\n DRAW MODE");
  printf("\n 0 = JAM1  1 = JAM2");
  printf("  2 = COMPLEMENT 4 = INVERSID\n");
  scanf("%ld",&dm);

/* Oeffnen des Ausgabewindows */
  font_window = (struct Window *)
                        OpenWindow(&windowdata);
  if(!font_window)ende();


```

```
/**************************************************
 * Style und Flags hier auf NULL gesetzt.         */

  textattr.ta_Style = (UBYTE)NULL;
  textattr.ta_Flags = (UBYTE)NULL;


/**************************************************
 * Eigentlich benutzt OpenDiskFont() auch den     *
 * Speicher, doch zur Verdeutlichung wird zuerst  *
 * mit OpenFont() geprueft ob der gewuenschte Font*
 * im Speicher liegt. Ist das nicht der Fall,     *
 * wird OpenDiskFont() versucht                   */

  if(tf = (struct TextFont *)OpenFont(&textattr))
    printf("\n %s ist im Speicher und benutzbar\n",
                                textattr.ta_Name);
  else
   {
    printf("\n im Speicher ist %s nicht !!",
                                textattr.ta_Name);
    printf("\n Schauen wir auf die Diskette.\n");

    if(tf = (struct TextFont *)
               OpenDiskFont(&textattr))
     printf
        ("\n UFF! in der FONTS: Directory gefunden");
    else
     {
      printf("\n Auch auf der Diskette kein %s",
                                 textattr.ta_Name);
      ende();
     }
    printf("\n\nADIEU !!\n");
   }

/***************************************************
 * Ist der Font gefunden wird er in mehreren Arten *
 * gezeigt.                                        */

  SetFont(RP,tf);

  SetDrMd(RP,dm);
  SetAPen(RP,2L);
  SetBPen(RP,3L);

  Move(RP,10L,25L);
  Text(RP,"abcde",5L);

  Move(RP,200L,25L);
  Text(RP,textattr.ta_Name,str_len((char *)textattr.ta_Name));

  SetSoftStyle(RP,FSF_ITALIC,255L);
  Move(RP,200L,40L);
  Text(RP,"abcde",5L);

  SetSoftStyle(RP,FSF_BOLD,255L);
  Move(RP,10L,40L);
  Text(RP,"abcde",5L);

  SetSoftStyle(RP,FSF_UNDERLINED,255L);
  Move(RP,100L,25L);
  Text(RP,"abcde",5L);

  SetSoftStyle(RP,FSF_BOLD|FSF_UNDERLINED,255L);
  Move(RP,100L,40L);
  Text(RP,"abcde",5L);

 /* Der Task gibt den Fontzugriff auf */
  CloseFont(tf);

 /* Das Programm wartet, bis das Close-Gadget des
      Windows angeklickt wird */
  while(font_window->MessageKey->Class != CLOSEWINDOW);

/***************************************************
 * Der Font wird wieder aus dem System entfernt.   *
 * Benutzt man diese Funktion nicht, so verbleibt  *
 * der gewuenschte Font, als Systemfont, im Speicher*
 * auch wenn dieses Progamm beendet ist.           *
 * Startet man das Programm dann wieder und verlangt*
 * denselben Font, so erhaelt man die Meldung, da_ *
 * sich dieser im Speicher befindet.               */

  RemFont(tf);
```

dener Font wird aus diesem entfernt. Übergeben wird der Zeiger auf die TextFont-Struktur. Im Fall AddFont() wurde der Speicherplatz für die Struktur mit AllocMem() reserviert, daher muß man diesen mit FreeMem() wieder freigeben, um den Font endgültig zu entfernen und den Speicher zurückzugewinnen. In den beiden anderen Fällen ist der Speicher nach RemFont() direkt wieder frei. Konnte der Font nicht entfernt werden, da er vielleicht gar kein Systemfont war, ist der Fehlerwert <TRUE>, also ungleich Null.

fehler = SetFont(rastport, textfont);

Ü: struct RastPort *rastport (A1).
struct TextFont *textfont (A0).
R: LONG fehler (D0).

Ein Systemfont wird einem RastPort zugewiesen.
Eine verfügbarer, also ein im RAM oder ROM vorhandener, TextFont wird auf einem RastPort aktiviert. Alle in Zukunft mit Text() auf dem RastPort ausgegebenen Zeichen werden in dem durch TextFont spezifizierten Font dargestellt.

neuerStil = SetSoftStyle(rastport, style,enable);

Ü: struct RastPort *rastport (A1).
UBYTE style (D0).
UBYTE enable (D1).
R: UBYTE neuerStil (D0).

Ändern des Darstellungsstils eines Fonts.
Die durch SetSoftStyle() erzeugten Änderungen beziehen sich auf die Styleflags. Man kann hiermit die Darstellunsart, bezogen auf einen RastPort, nachträglich ändern. Es gibt jedoch Einschränkungen. Ist nämlich in der betreffenden TextFont-Struktur, welche mit dem RastPort zusammenarbeitet, ein Style schon fest vorgegeben, läßt sich dies nicht rückgängig machen oder gar verdoppeln. Um dies zu kontrollieren, braucht die

Funktion eine Kontrollmaske, in der sie erkennen kann, was wirklich verändert werden darf und was nicht. Diese wird durch die Variable <enable> dargestellt. Wir erhalten diese Variable von der Funktion AskSoftStyle(), die also zuvor aufgerufen werden muß. Den neuen Zustand des auf dem RastPort gültigen Fonts erhalten wir in <neuerStil>, was dem <ta_Style>-Element in TextAttr entspricht.

fehler = Text(rastport,string, stringlänge);

Ü: struct RastPort *rastport (A1).
STRPTR string (A0).
LONG stringlänge (D0).
R: LONG fehler (D0).

Eine Zeichenkette wird gezeigt.
Hiermit können wir nun endlich die Früchte unserer mühsamen Arbeit ernten. Ein String wird in dem im RastPort gültigen Font, dessen Style und dessen DrawMode auf den Bildschirm gebracht. Die Stringlänge ist die Anzahl der Zeichen im String, weswegen dieser, entgegen den ansonsten in C üblichen Konventionen, nicht mit einem Nullbyte abgeschlossen werden muß. Die Ausgabe wird mit Hilfe der Funktion Move() positioniert, welche im Artikel "Grafik mit dem Betriebssystem" beschrieben wird. Die durch Move() bestimmte Position ist die Baseline des darzustellenden Fonts. Außerdem kann man die im selben Artikel beschriebene Funktion SetDrMd() gleichsam für Text() anwenden.

pixellänge= TextLength (RastPort, string, stringlänge

Ü: (siehe Text()).
R: LONG pixellänge (D0).

Die Übergaben sind mit denen der Funktion Text() identisch, jedoch erfolgt keine Bildschirmausgabe, denn TextLength() gibt nur Auskunft darüber, wieviele Pixel der Text bei der Ausgabe in der Horizontalen einnimmt. Je nach benutztem Font kann diese Länge sehr unterschiedlich sein.

```

/* Probieren sie es aus !!!                                   *
 ************************************************************/

  ende();
  return(NULL);
}

LONG str_len(s)
 char *s;
 {
  char *cp = s;

  while(*cp++);
  return(cp-s-1);
 }

ende()
{
 if(GfxBase)          CloseLibrary(GfxBase);
 if(IntuitionBase)    CloseLibrary(IntuitionBase);
 if(DiskfontBase)     CloseLibrary(DiskfontBase);
 if(font_window)      CloseWindow(font_window);
 return(NULL);
}

                  --- END OF FILE : showfont.c ---
========================================================================
========================================================================
LISTING  VON :                                            Seiten ____
listfonts.c                       2811 rwed Today       17:00:49
================================================================ 3.4 a ======

/* Dieses Programm listet alle verfuegbaren Fonts
 * in ihren Hoehen auf und zeigt wo diese herkommen.
 * Es zeigt die Funktionsweise von AvailFonts().
 ******************************************************/

#include <intuition/intuitionbase.h>
#include <libraries/diskfont.h>

/* nur fuer Actec C */
#include <functions.h>

#include <exec/memory.h>
#include <libraries/dos.h>

/* Groesse des Buffers fuer AvailFonts() */
#define FNAMEBUF 1000L

struct DiskfontBase     *DiskfontBase;
struct IntuitionBase    *IntuitionBase;
struct GfxBase          *GfxBase;

struct AvailFontsHeader *afh;
struct AvailFonts *af;

main()
 {
  LONG err;
  SHORT i;

  if( (DiskfontBase = (struct DiskfontBase *)
        OpenLibrary("diskfont.library",0L)) == NULL )
              ende();

  if( (IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library",0L)) == NULL )
              ende();

  if( (GfxBase = (struct GfxBase *)
        OpenLibrary("graphics.library",0L)) == NULL )
              ende();

/*****************************************************
 * Reservieren eines Speicherbereichs fuer den Buffer*
 * von AvailFonts                                     */

  afh = (struct AvailFontsHeader *)
             AllocMem(FNAMEBUF,MEMF_CLEAR);
  if(!afh) ende();


/*****************************************************
 * Der Aufruf selbst. Durch Uebergabe von AFF_DISK    *
```

```
 * und AFF_MEMORY, wird bewirkt, sowohl im Speicher  *
 * als auch in FONTS: gesucht wird.                  */

 err = AvailFonts(afh,FNAMEBUF,AFF_DISK|AFF_MEMORY);
 if(err)
  {
   Write(Output(),"AVAIL FONTS ERR\r\n",16L);
   ende();
  }


/*****************************************************
 * Nun wird ueber die AvailFontsHeader Struktur      *
 * gesprungen, und ein Zeiger auf die folgenden      *
 * AvailFonts Strukturen gebildet.                   */

 af = (struct AvailFonts *)&afh[1];


/*****************************************************
 * Es werden alle Eintraege durchgeschaut und        *
 * angezeigt.                                        */

 for( i = 0; i < afh->afh_NumEntries; i++)
  {
   printf("\n Font : %s",
                     (af+i)->af_Attr.ta_Name);
   printf("\n mit einer Hvhe von %u Pixeln",
                     (af+i)->af_Attr.ta_YSize);

   if( (af+i)->af_Type == AFF_DISK )
    printf("\n in FONTS: gefunden\n");
   else
    printf("\n im Speicher gefunden\n");
  }

 ende();
 return(NULL);
}

/*******************************************
 * Schliessen der geoeffneten Librarys und*
 * Speicherbereiche                       */

ende()
{
 if(GfxBase)          CloseLibrary(GfxBase);
 if(IntuitionBase)    CloseLibrary(IntuitionBase);
 if(DiskfontBase)     CloseLibrary(DiskfontBase);
 if(afh)              FreeMem(afh,FNAMEBUF);
 return(NULL);
}

                    --- END OF FILE : listfonts.c ---
================================================================
```

Nun aber zur eigentlichen Beschaffenheit eines Fonts. Wie zu Beginn schon erwähnt, handelt es sich dabei um ein Bitfeld. Dieses Bitfeld hat eine Höhe von sovielen Bits, wie durch <YSize> festgelegt wurde. Diesen Wert <YSize> finden wir in allen Strukturen, die etwas über ein Font aussagen. Vor weiteren Überlegungen schauen wir uns die einzelnen Komponenten der Struktur TextFont an, die einen Font in allen Einzelheiten definiert.

struct Message tf_Message;

Eine Exec-Struktur, die zur Verbindung des Fonts mit dem System dient.

UWORD tf_YSize;

Die Höhe des Fonts in Pixel (=Bits), also auch die Höhe des besagten Bitfeldes.

UBYTE tf_Style;

Der Style des Fonts. Hier sind die Flags mit dem Beginn <FS_> bzw. <FSF_> einzusetzen. Wird hier ein Flag gesetzt, so bleibt dieses Attribut fest mit dem Font verbunden. Wird <FS_NORMAL> gesetzt, kann man die anderen Attribute später mit SetSoftStyle() aktivieren.

UBYTE tf_Flags

Die hier zu setzenden Flags beginnen mit <FPF_>. Bei der Erstellung eigener Fonts sollte man, den Empfehlungen der Amiga-Entwickler nach, auf <FPF_ROMFONT> verzichten, es sei denn, man befasse sich mit dem Gedanken, neue ROMs zu brennen. Es müßte aber möglich sein, das KickStart der 1000er Amigas zu mo-difizieren.

UWORD tf_XSize;

Hier wird die normale Breite eines Zeichens in Pixeln festgehalten.

UWORD tf_BaseLine;

Die BaseLine ist die von oben abgezählte Bitreihe, in der sich der unterste Punkt eines Zeichens befindet. Darunter kann dann noch Platz für die Unterstreichung bleiben oder z.B. der unterste Punkt eines "y","g" etc. dargestellt werden.

UWORD tf_BoldSmear;

Wird hier Eins eingesetzt, können die Zeichen fett gedruckt werden.

UWORD tf_Accessors;

Dies ist nur ein Zugriffszähler, der auf Null gesetzt werden sollte. Man kann hier später abfragen, ob und wieviele Tasks auf den Font zugreifen.

UBYTE tf_LoChar;

Normalerweise kann ein Font aus 256 Zeichen bestehen. Wem das zuviel ist, da diese Menge nicht immer benötigt wird, der muß hier den unteren ASCII-Wert, der dann dem ersten eigenen Zeichen entspricht, angeben. Die weiteren definierten Zeichen folgen dann der Reihe nach mit entsprechenden ASCII-Werten bis zu:

UBYTE tf_HiChar;

...welches dann den ASCII-Wert des letzten definierten Zeichens darstellt.

APTR (unsigned Char *) tf_CharData;

Der Zeiger auf das vielzitierte Bit-feld. Die Daten entsprechen den reihenweisen Bitfolgen, zuerst die obere Reihe, dann die zweite von oben usw., immer von links nach rechts. Nach dem letzten Zeichen muß man noch ein weiteres definieren, das gezeigt wird, wenn dem gegebenen ASCII-Wert kein entsprechendes Fontzeichen zugeordnet ist.

UWORD tf_Modulo;

Dieser Wert stellt die Breite des Bitfeldes in Byte dar, also Breite in Pixel geteilt durch 8. Hätte man also einen Zeiger auf ein Bit einer bestimmten Spalte, und würde auf diesen Zeiger <tf_Modulo> addieren, hätte man einen Zeiger auf ein Bit derselben Spalte, in der nächsten Reihe.

APTR tf_CharLoc;

Ein Zeiger auf ein Datenfeld, in welchem sich für jedes Zeichen zwei WORDs bereithalten. Das erste WORD gibt die Spaltenzahl an, in der das Zeichen beginnt, das zweite die Bitbreite des Zeichens. Das geschieht in der Reihenfolge: Spalte (1.Zeichen),Breite(1.Zeichen), Spalte (2.Zeichen),usw.. Dieses Datenfeld ist für die Erstellung proportionaler Zeichensätze wichtig.

APTR tf_CharSpace;

Ein Zeiger auf ein WORD-Datenfeld, in dem nacheinander die Anzahl der freien Spalten nach einem Zeichen stehen. Man kann sich allerdings auch auf die Gesamtzeichenbreite <tf_X-Size> beziehen; dazu ist dieser Zeiger auf Null zu setzen.

APTR tf_CharKern;

Ein weiterer Zeiger auf ein weiteres Datenfeld. In diesem befinden sich WORDs, die die Anzahl der freien Spalten innerhalb des Bitbereichs eines Zeichens vor dem ersten gesetzten Bit. darstellen.

Wir haben also nicht nur die Möglichkeit, mittels <tf_XSize> eine starre Zeichenbreite zu erzeugen, sondern können auch variable Zeichenbreiten innerhalb eines Fonts benutzen. Dies nennt man Proportionalschrift. Der Effekt ist, daß ein kleines “i” dann nicht mehr soviel Raum einnimmt wie z.B. ein großes “M”. Hat man eine solche TextFont-Struktur komplett initialisiert, so kann man diese an AddFont() übergeben und den erstellten Font somit benutzen. Theoretisch sollte nun alles klar sein. Um sich den praktischen Ablauf zu verdeutlichen, sollte das Listing addmyfont.c näher begutachtet werden.

Man kann auch eigene Diskfonts erzeugen. Ein Beispiel dafür ist das Assembler-Listing makediskfont.asm. Lassen Sie sich nicht durch den Namen Assembler abschrecken, denn es sind keinerlei Kenntnisse darüber erforderlich. Sie müssen nur die besprochenen TextFont-Daten an den jeweils gekennzeichneten Stellen in das Listing einfügen und dieses dann assemblieren und linken. Weiterhin braucht man noch einen File, damit das System Ihren eigenen Diskfont findet. Um sich zu verdeutlichen, wie Diskfonts aufgebaut sind, schauen Sie doch einmal in eine “font”-Directory.

Diese sieht etwa so aus:

```
opal (dir)
ruby (dir)
garnet (dir)
opal.font   ruby.font
garnet.font
```

Die xxx.font-Files sind die besagten Files mit den Verwaltungsinformationen. Da man einen Font ja in diversen Größen definieren kann, steht in diesem File, wieviele Definitionen, also welche Größen vorhanden sind, und wo die eigentliche Definition zu finden ist. Diese sollte immer in der “fonts”-Directory zu finden sein. Sie sehen in der Directory “fonts” mehrere Subdirectories mit den entsprechenden Namen. In diesen befinden sich die eigentlichen Diskfont-Files, die einfach nur ihre jeweilige Höhe zum Namen haben. Um diesen Standard einzuhalten, kreieren Sie Ihre Diskfonts nach folgendem Schema:

Angenommen: Man hat einen Font “myfont” in zwei Größen entworfen, nämlich 9 und 14.

Eröffnen Sie im Verzeichnis “fonts” ein Unterverzeichnis “myfont”.

MAKEDIR fonts:myfont

Nun kopieren Sie die mit makediskfont.asm erstellten Files 9 und 14 in das eröffnete Verzeichnis.

Starten Sie nun das Programm makediskfontfile und beantworten Sie dessen Eingabeanforderungen. Den dadurch entstandenen File “myfont.font” kopieren Sie in das “fonts”-Verzeichnis.

Jetzt ist der neue Font verfügbar und kann aus Programmen wie Notepad oder Vizawrite benutzt werden. Sie können ihn auch mit dem Programm showfont anzeigen lassen oder mit listfonts sein Vorhandensein überprüfen.

Der Vorteil der Diskfonts ist, daß sie sich von jedem Programm aus nutzen lassen, das mit Fonts arbeitet. Man kann natürlich auch, bevor man ein solches Programm startet, einen eigenen Font mit addmyfont in das System integrieren. Doch Vorsicht! Einen Reset überlebt dieser Font nicht.

Nun sollte es Ihnen möglich sein, Fonts in allen erdenklichen Weisen zu nutzen. Damit und bei Anderem wünsche ich Ihnen viel Spaß und wenig Gurus.

Folgende Programmlistings sollen beim Arbeiten mit Fonts Unterstützung liefern.

1. showfont.c (zeigt einen beliebigen vorhanden Font)
2. listfonts.c (zeigt alle verfügbaren Fonts und deren Eckdaten)
3. addmyfont.c (definiert einen Font und bindet ihn in das System ein)
5. makediskfont.asm und
6. makediskfontfile.c (dienen zur Erstellung eines Diskfonts)

Hinweise zur Benutzung und Erstellung der Programme finden Sie in den Listings selbst.

VON FRANK SCHÄFER

DER COPPER

Schon lange geistert dieses Bauteil des Amiga durch die Presse und durch die Gespräche der "Fachkundigen ". Wir wollen nun etwas Klarheit in die BlackBox des Agnus bringen, dort ist der Copper nämlich zu finden.

Wie viele sicher wissen, ist der Agnus ein VLSI- Bauteil (Very Large Scale Integration) wie auch die beiden anderen Custom-Chips. Der Copper ist der springende Punkt der Amigagrafik. Er ist ein echter Coprozessor, der parallel zum 68000 arbeitet, was bedeutet, daß er einen eigenen DMA-Kanal (Direct Memory Access) zur Verfügung hat. Er läuft synchron zum Elektronenstrahl des Monitors und kann abhängig von dessen Position spezielle Operationen durchführen, zum Beispiel das Umschalten von Farbtabellen (mehrere Screens mit verschiedenen Farben) oder die Änderung der Auflösung (mehrere Screens mit verschiedenen Auflösungen). Da der Copper ein Coprozessor des 68000 ist, besitzt er logischerweise auch einen eigenen Befehlssatz. Programmteile (Copper Instruction List), die auf den Copperbefehlssatz zugreifen, werden abgearbeitet, während die CPU ihre eigenen Befehle abarbeitet (68000 Instruction List). Erst durch diese Technik ist es überhaupt möglich, mit vielen Screens zu arbeiten. Diese parallele Verarbeitung ist natürlich nicht wirklich parallel, da der Bus, über den Speicherzugriffe laufen, nur eine Informationseinheit gleichzeitig übertragen kann. Doch hier kommt eine weitere Stärke des Amiga zum Tragen. Der Systemtakt liegt nämlich nicht bei 7.19 MHz, sondern bei 14.38 MHz. Diesen Takt "teilen" sich die CPU und die Costumchips. Für jeden Chip sieht es dann so aus, als würde das System mit 7.19 MHz arbeiten. Erst durch diesen hohen Systemtakt wird der Amiga so schnell, wie er ist. Während des Einschaltens des Amiga ist der Copper erst einmal inaktiv, d.h nur der MC68000 ist aktiv. Erst das Betriebssystem des Amiga teilt der CPU mit, daß doch bitte der Copper zu aktivieren sei. Das Betriebssystem, in Form der CPU, schreibt hierzu extra ein paar Copper-Befehle in den Speicher. Die CPU teilt nun dem Copper über ein spezielles Hardwareregister mit, wo er diese Instruktionen vorfindet. Danach hat die CPU nichts mehr mit der Kontrolle des Videodisplays zu tun, diese Tätigkeit hat der Copper nun vollständig übernommem.

Die Softwareseite des Coppers

Dem Programmierer stehen, wie schon erwähnt, spezielle Befehle zur Coppersteuerung zu Verfügung. Mit diesen Befehlen ist es möglich, die Effekte zu erzeugen, die man u.a. in den Bootintros bewundern kann. Doch nun zum Copper-Befehlssatz. Obwohl dieser Befehlssatz nur drei (!) Instruktionen beinhaltet, ist es mit ihm möglich, all das zu machen, was oben beschrieben wurde. Diese drei Befehle sind WAIT, MOVE und SKIP. Sie umfassen immer ein LongWord (4 Byte, 32 Bit), und der Copper greift grundsätzlich auf das komplette LongWord zu. Der Copper hat weiterhin Zugriff auf fast alle Hardware-Register des Amiga. Durch eine Veränderung dieser Register wird der Copper und damit das Videodisplay gesteuert. Doch das Betriebssytem des Amiga stellt dem Benutzer außer dem Wait-, Move- und Skip-Befehl noch einige Makros zur Copperprogrammierung zur Verfügung, als da wären:

CEnd

Dieser Befehl beendet eine vom Programmierer erstellte Copperlist.Die Syntax lautet:

CEnd (c)

c : Ein Pointer (Zeiger) auf eine UCopList.
Er ruft zuerst CWait auf, und zwar so, daß der Copper weiß, hier ist die Liste zu Ende, und setzt dann den Pointer durch Aufruf von CBump auf die nächste Speicherstelle, wo man anschließend eine weitere Instruktion eintragen könnte.

CInit

Mit diesem Befehl kann man den Kopf einer beliebigen Copperlist aufbauen. Die Syntax lautet:

CInit (c, n)

c : Ein Pointer (Zeiger) auf eine UCopList.
n : Die Anzahl der Instruktionen.

CInit belegt den benötigten Speicher für n Instruktionen und für den Pointer der Copperlist. Falls c schon definiert ist, wird die Datenstruktur der Liste neu initialisiert. Man kann anschließend am Ende der Liste neue Befehle anhängen. CINIT ruft, um dies zu machen, u.a. UCopperListInit auf.

CMOVE

Hängt einen Move-Befehl an das Ende der aktuellen Liste an. Die Syntax lautet:

CMOVE (c, a, v)

c : Wie immer der Pointer zu der aktuellen UCopList.
a : Das betreffende Hardwareregister.
v : Die 16-Bit- Zahl, die in in die Liste eingetragen werden soll.
Grundsätzlich ruft CMOVE zuerst CMove auf, anschließend wird CBump aufgerufen. Dadurch wird der Pointer auf den nächsten Befehl gesetzt.

CWAIT

Wartet, bis der Elektronenstrahl die angegebene Position erreicht hat. Die Syntax lautet:

CWAIT (c, y, x)

c : Nun ja, das gleiche wie immer.
y : Die Y-Position, bis zu der der Copper warten soll.
x : Die X-Position, bis zu der gewartet werden soll.
Auch CWAIT ist wieder ein Makro, zuerst wird CWait, anschließend C-Bump aufgerufen. Das Resultat ist, daß der Copper erst dann, wenn die angegebene Position erreicht ist, die nächste Instruktion ausführt.

UCopperlistInit

Baut den Kopf einer User-Copperlist auf. Syntax:

UCopperList (c, n)

c : Pointer zu einer UCopperlist.
n : Anzahl der folgenden Instruktionen.
(Siehe: CINIT)Außer den reinen Coppersteuerbefehlen gibt es noch Befehle zur Copperlist-Behandlung. Mit ihnen kann man den von der Copperlist

AZTEC-C FÜR AMIGA VERSION 3.6

Wußten Sie, daß eines der verbreitesten und komplexesten Betriebssysteme - UNIX - in C geschrieben ist ?

NEU!
V.3.6

SOFTWARE DEVELOPMENT SERIES
AZTEC
C
MANX

Wußten Sie, daß auch das Betriebssystem des AMIGA größtenteils in C geschrieben wurde ?

Wußten Sie, daß C eine der wichtigsten und modernsten Programmiersprachen ist ?

Wußten Sie, daß Aztec-C einer der schnellsten und leistungsfähigsten Compiler für den AMIGA ist ?

Wußten Sie, daß jetzt Aztec-C in der Version 3.6 verfügbar ist ?

Möchten Sie mehr darüber wissen? Dann schicken Sie uns einen ausreichend frankierten Rückumschlag und Sie erhalten ausführliche Information.

Up-Date-Service für alle MANX-Kunden auch bei uns. Fragen Sie nach!

HIERMIT BESTELLE ICH:

- ☐ AZTEC-C68K/AM-P PROFESSIONAL SYSTEM FÜR DM 398.-
- ☐ AZTEC-C68K/AM-D DEVELOPER SYSTEM FÜR DM 598.-
- ☐ AZTEC-SDB SOURCE LEVEL DEBUGGER FÜR DM 149.-

Versandkosten: Inland DM 7,50 Ausland DM 10,-
Auslandbestellungen nur gegen Vorauskasse
Nachnahmegebühr DM 3,70

- ☐ Vorauskasse
- ☐ Nachnahme

NAME: ____________________

VORNAME: ____________________

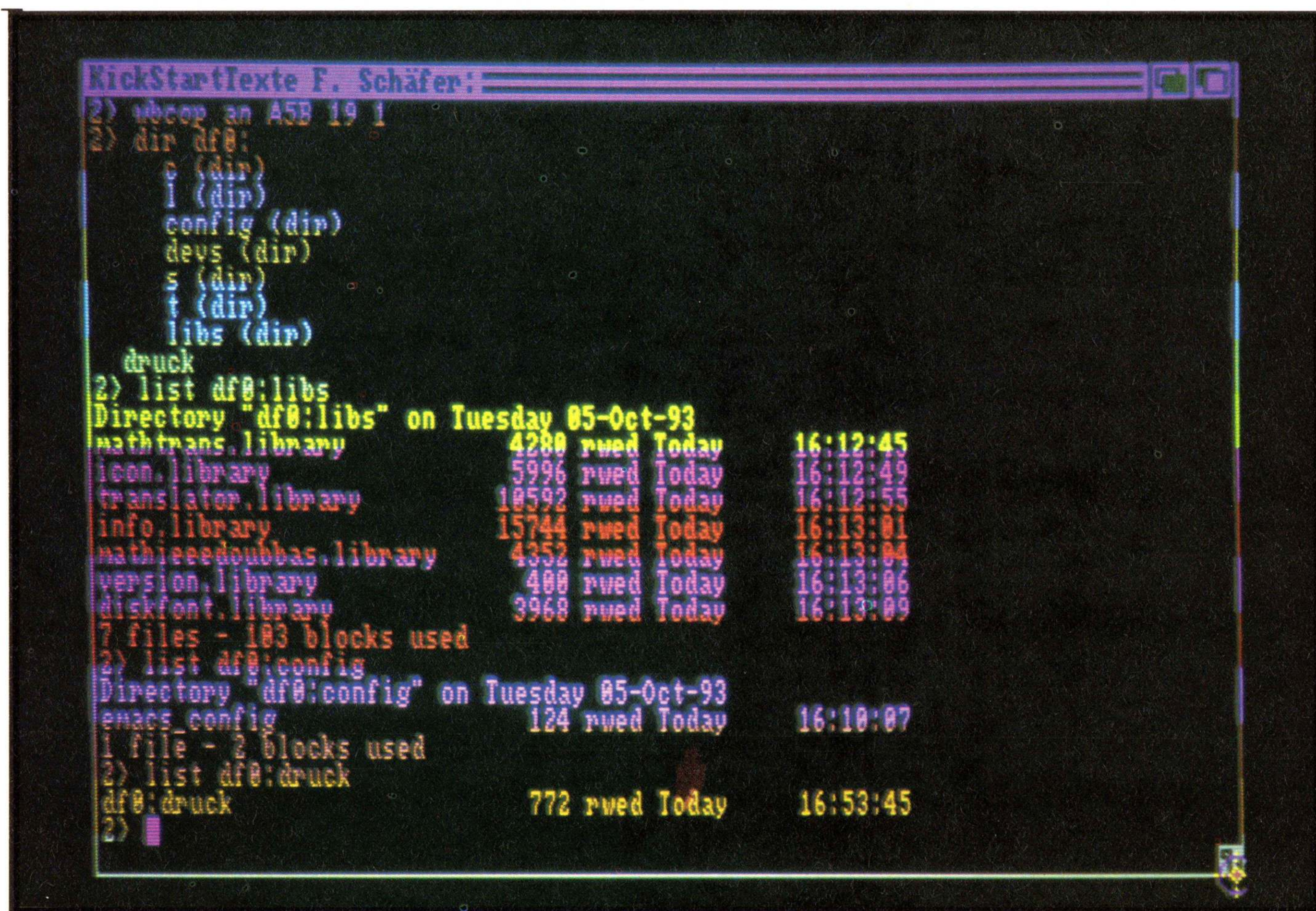
STRAßE: ____________________

ORT: ____________________

UNTERSCHRIFT: ____________________



MERLIN COMPUTER GMBH
INDUSTRIESTRAßE 26
6236 ESCHBORN
TEL. 06196/481811



Auch ein buntes Schriftbild läßt sich erzeugen.

belegten Speicher an das System zurückgeben. Da wir gerade beim Speicher sind, dazu noch ein paar Anmerkungen. Der Copper kann, wie die anderen Customchips auch, nur auf CHIP-Memory zugreifen. Da das Betriebssystem aber grundsätzlich FAST-Memory (falls davon etwas da ist) belegt, ist mit Problemen zu rechnen, wenn man beim Speicherbelegen nicht angibt, welche "Sorte" RAM man haben will. Viele der sogenannten Cracker berücksichtigen dies nicht, und logischerweise stürzen ihre Programme dann ab, wenn man sie auf einem Amiga mit FAST-Memory startet. Doch jetzt zu den angesprochenen Prozeduren.

```
FreeCopList
```

Gibt den Speicher einer Usercopperlist an das System zurück. Syntax:

```
FreeCopList ( c )
```

c : Pointer zu einer CopList.

```
FreeCprList
```

Gibt den Speicher einer Hardwarecopperlist an das System zurück. Syntax:

```
FreeCprList ( c )
```

c : Pointer zu einer Cpr List.

Der nützlichste von allen Befehlen ist, obwohl man es kaum glauben mag, der Wait-Befehl. Durch eine direkte Kontrolle der Hardware-Register ist es dem Copper möglich, zu warten, bis der Elektronenstrahl auf dem Bildschirm eine spezielle Position erreicht hat. Wenn dies der Fall ist, werden die für diesen Punkt spezifizierten Befehle ausgeführt, zum Beispiel ein MOVE-Befehl. Mit diesem MOVE-Befehl wird Datentransfer innerhalb des Speichers bewältigt. Man kann mit ihm die Copperlist aufbauen und manipulieren. Da der Zugriff des Coppers direkt auf die Register erfolgt, wird der Amiga praktisch nicht verlangsamt. Nach einem Wait-Befehl können übrigens mehrere Move-Befehle stehen. Mit dem Skip-Befehl ist dann noch möglich, Schleifen bzw. GoTo-Operationen zu programmieren.

Die Tätigkeiten des Coppers

Wenn nur die Workbench oder ein einzelner Screen auf dem Bildschirm zu sehen ist, hat der Copper nicht viel zu tun. Anders wird das, wenn man mehrere Screens auf dem Bildschirm hat; der Copper wird dann stark beansprucht. Wenn, wie meist, die Auflösungen und die Farbtabellen der Screens völlig verschieden sind, kommen die Fähigkeiten des Coppers zum Tragen. Er erhält von der CPU die Meldung "Achtung! an der betreffenden Stelle bitte auf eine andere Farbtabelle und Auflösung umschalten". Der Copper wartet, und wenn der Elektronenstrahl an der festgelegten Stelle ist, schaltet er um. Allerdings kann er nicht an jeder beliebigen Stelle und auch nicht beliebig oft schalten. In der Horizontalen kann

der Copper maximal 113mal schalten, das sind in LoRes-Auflösung 4 und in HighRes-Auflösung 8 Punkte. In der Vertikalen sind an sich 255 Schaltpositionen möglich. In dem Beispielsprogramm werden allerdings keine Umschaltungen auf horizontalen Positionen vorgenommen. Wenn man dies machen würde,wäre es möglich, im LoRes-Modus mehrere hundert Farben darzustellen.

Doch nun zur Erklärung des Copperprogramms. Wie man auf dem Foto sieht, ist auf der Workbench eine von oben nach unten verlaufende Färbung zu sehen. Diese Färbung kommt dadurch zustande, daß der Copper an 16 Y-Positionen auf eine andere Färbung umschaltet.

Aufruf: WBCop (an|aus) [Farbe (0..FFF)] [Differenz (+- 0..F0)] [Farbnummer (0..3)]

Farbe : Farbe, mit der angefangen werden soll, Angabe in Hex-Werten.

FFF = weiß
F00 = rot
0F0 = grün
00F = blau

Wie man sieht, kann jede Farbe in 16 (0 .. F) verschiedenen Intensitäten dargestellt werden. Durch Mischen der Farben kann man 16 * 16 * 16 = 4096 verschiedene Intensitäten darstellen.

Differenz : Differenz zwischen zwei Farben, Angabe in Hex-Werten. Es können positive oder negative Differenzen angegeben werden.

Farbnummer :
Farbnummer 0 = Hintergrundfarbe
Farbnummer 1 = Textfarbe
Farbnummer 2,3 = Die beiden anderen Farben der Workbench

Wenn man nur "WBCop an" eingibt, wird die voreingestellte Copperlist auf die Workbench gelegt (siehe CONST im Kopf des Programmes). Bei " WBCop an F10 1 1 " wird eine von Rot nach Violett übergehende Copperlist auf die TextColor der Workbench gelegt. Mit "WBCop aus" wird die aktuelle Copperlist wieder ausgeschaltet und der belegte Speicher ans System zurückgegeben.

Voraussetzung für das Einblenden einer Copperlist ist, daß man die ViewPort-Adresse des betreffenden Screens, hier der Workbench, kennt, denn im ViewPort ist der Pointer auf die aktuelle Copperlist des Screens definiert. An den ViewPort der Workbench heranzukommen, ist im Mayer-Vogt Modula-2 recht einfach. Man ruft einfach die Prozedur OpenWorkBench auf; diese Prozedur gibt den Screen-Pointer der WorkBench zurück. Da in der Screen-Struktur ein Pointer auf den ViewPort existiert, kann man dann leicht die Adresse des ViewPorts auslesen (durch die Kenntnis des Screen-Pointers kann man Operationen vornehmen, wie Einblendungen in die Screen-Titelzeile (Pfad-Namen, Uhrzeit etc.)). Wenn man den Pointer auf den Workbench-ViewPort ausgelesen hat, wird anschließend die aktuelle Copperlist gelöscht. Das geschieht mit der Prozedur FreeCopList. Falls mehrere Copperlisten vorhanden sind, werden alle gelöscht. Nun wird mit CWait, CBump und CMove in der Prozedur WBCopper eine neue Copperlist aufgebaut und auf den betreffenden Screen gelegt. Wenn WBCop mit dem Parameter <aus> aufgerufen wird, löscht die Prozedur ClearCopperList die aktuelle Copperlist, anschließend wird eine "leere" Copperlist auf den Screen gelegt. Falls weder die Option <an> noch die Option <aus> angewählt wurde, bekommt der Benutzer eine Kurzanleitung des Programms zu sehen. And now, have a lot of fun with your CopperWorkbench!

```
MODULE WBCop;

          (* Copper-Steuerung von F. H. Schaefer *)
          (* If YOU find a bug : call Germany 06035 / 4439 *)
          (* Auf AMIGA 1000 mit 2.5 MB und MC68010 entwickelt *)
          (* Laeuft aber auf jedem Amiga *)

FROM Arguments   IMPORT GetArg, NumArgs;
FROM Arts        IMPORT Assert;
FROM Conversions IMPORT StrToVal;
FROM Exec        IMPORT MemReqs, MemReqSet, AllocMem;
FROM Graphics    IMPORT UCopListPtr, ViewPortPtr, CBump, CMove, CWait,
                        FreeCopList;
FROM Hardware    IMPORT custom;
```

```
FROM InOut          IMPORT WriteString, WriteLn;
FROM Intuition     IMPORT ScreenPtr, RethinkDisplay, OpenWorkBench,
                   DisplayAlert;
FROM Storage       IMPORT ALLOCATE;
FROM Strings        IMPORT Compare;
FROM SYSTEM        IMPORT ADDRESS, ADR;


(*---------------------------------------------------------------------------*)


CONST DefaultColor          = 0F0H;
      DefaultDifferece      = 1;
      DefaultColorNumber = 0;

(*---------------------------------------------------------------------------*)

TYPE  STRING = POINTER TO ARRAY [1..256] OF CHAR;

(*---------------------------------------------------------------------------*)

VAR Text           : STRING;
    WBVPort                : ViewPortPtr;
    Laenge                 : INTEGER;
    cl                     : UCopListPtr;
    Color,Difference,
    ColorNumber            : LONGINT;
    Vorzeichen,Error       : BOOLEAN;

(*---------------------------------------------------------------------------*)

PROCEDURE NoMemory;

(* Darstellen einer Systemmeldung mit einem Guru-Alert *)

VAR
 AlertText: RECORD
          XPosition1      : CARDINAL;
          Text1           : ARRAY [0..59] OF CHAR;
          XPosition2      : CARDINAL;
          Text2           : ARRAY [0..12] OF CHAR
 END;

BEGIN
  WITH AlertText DO
    Text1      :="*Operator Failure.   Press left mouse button to continue. ";
    XPosition1 :=100;
    Text1[0]   :=CHAR (15);
    Text1[59]  :=CHAR (1);
```

```
    Text2       :="* No Memory";
    XPosition2 :=260;
    Text2[0]    :=CHAR (30);
    Text2[12]   :=CHAR (0)
  END;
  IF DisplayAlert (0, ADR(AlertText), 60) THEN END
END NoMemory;

(*-----------------------------------------------------------------------*)

PROCEDURE WBViewPort(VAR WBVP : ViewPortPtr );

VAR  WBScreenPtr : ScreenPtr;

BEGIN
   WBSc~eenPtr := OpenWorkBench ();
   WBVP := ADR (WBScreenPtr^.viewPort)
END WBViewPort;

(*-----------------------------------------------------------------------*)

PROCEDURE ClearCopperList (VAR WBVPort : ViewPortPtr);

VAR Copperlist   : UCopListPtr;

BEGIN
  WBViewPort (WBVPort);
  IF WBVPort # NIL THEN
      Copperlist:= AllocMem (SIZE (Copperlist^),
                   MemReqSet{chip, public,memClear});
      Copperlist := WBVPort^.uCopIns;
      WHILE Copperlist # NIL DO
         FreeCopList (Copperlist^.firstCopList);
         Copperlist := Copperlist^.next
      END
  ELSE
    NoMemory
  END
END ClearCopperList;

(*-----------------------------------------------------------------------*)

PROCEDURE WBCopper (ViewPort : ViewPortPtr;
                 Color,Difference,ColorNumber : INTEGER);

VAR i     : INTEGER;
    cl    : UCopListPtr;

```

```
BEGIN
  cl:=AllocMem (SIZE(cl^), MemReqSet{chip, public,memClear});
  IF cl # NIL THEN
     FOR i:=0 TO 15 DO
        CWait (cl,i*16,0);
        CBump (cl);
        CMove (cl, ADR (custom.color[ColorNumber]), Color + Difference * i);
        CBump (cl)
     END;
     CWait (cl, 0FFFH, 0FFEH);
     CBump (cl);
     ViewPort^.uCopIns := cl;
     RethinkDisplay
  ELSE
     NoMemory
  END
END WBCopper;

(*-----------------------------------------------------------------------*)

PROCEDURE Copperlist(Color,Difference,ColorNumber : INTEGER);

VAR  WBVPort     : ViewPortPtr;

BEGIN
 Assert ((Color < 4096) = TRUE, ADR (" Max 4096 (FFF) Farben"));
 Assert ((Difference < 0F0H) OR (Difference > -0F0H) = TRUE,
          ADR ("Nicht existente Farbe"));
 Assert ((ColorNumber < 4) = TRUE, ADR ("Nur Farben 0-3"));
 Assert ((Color + 16 * Difference >= 0) AND
        (Color + 16 * Difference < 4095) = TRUE,
         ADR ("Nicht existente Farbe"));
 ALLOCATE (WBVPort,SIZE(WBVPort^));
 IF WBVPort # NIL THEN
    ClearCopperList (WBVPort);
    IF WBVPort # NIL THEN
       WBCopper (WBVPort,Color,Difference,ColorNumber)
    END
 ELSE
    NoMemory
 END;
END Copperlist;

(*-----------------------------------------------------------------------*)

BEGIN
 ALLOCATE (Text,SIZE(Text^));
 GetArg (1,Text^,Laenge);
```

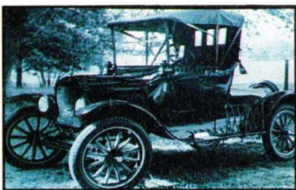
```
  IF Compare (Text^,0,3,"an",FALSE) = 0 THEN
    IF NumArgs () > 1 THEN
      GetArg (2,Text^,Laenge);
      StrToVal (Text^,Color,Vorzeichen,16,Error);
      Assert (Error AND Vorzeichen = FALSE, ADR ("Argument Fehler 1"));
      GetArg (3,Text^,Laenge);
      StrToVal (Text^,Difference,Vorzeichen,16,Error);
      Assert (Error = FALSE, ADR ("Argument Fehler 2"));
      GetArg (4,Text^,Laenge);
      StrToVal (Text^,ColorNumber,Vorzeichen,10,Error);
      Assert (Error AND Vorzeichen = FALSE, ADR ("Argument Fehler 3"));
      Copperlist (INTEGER(Color),INTEGER(Difference),INTEGER(ColorNumber))
    ELSE
      Copperlist (DefaultColor,DefaultDifferece,DefaultColorNumber)
    END
  ELSIF Compare (Text^,0,3,"aus",FALSE) = 0 THEN
    ALLOCATE (WBVPort,SIZE(WBVPort^));
    ClearCopperList (WBVPort);
    IF WBVPort # NIL THEN
      cl:=AllocMem(SIZE (cl^), MemReqSet{chip, public, memClear});
      IF cl # NIL THEN
        CWait (cl, 0FFFH, 0FFEH);
        CBump (cl);
        WBVPort^.uCopIns := cl;
        RethinkDisplay
      ELSE
        NoMemory
      END
    END
  ELSE
    WriteString (" Syntax : WBCop an [Farbe (0..FFF)] [Differenz (+-0..F0)]");
    WriteString (" [Farbnummer (0..3)]");
    WriteLn;
    WriteString (" Schaltet die Copperlist an ");
    WriteLn;
    WriteString (" Syntax : WBCop aus");
    WriteLn;
    WriteString (" Schaltet die Copperlist aus");
    WriteLn;
    WriteString(" Nur vom CLI aus benutzen");
    WriteLn;
    WriteString(" Von Frank Schaefer fuer Kickstart | Grafiksonderheft ");
    WriteLn
  END
END WBCop.

                    --- END OF FILE : wbcop.mod ---
================================================================
```

BILDERSHOWS:

Schon gleich nach dem Erscheinen des AMIGA in der Computerscene gab es die ersten Bildershows, die zeigten, was dieser Rechner zu leisten vermag. Künstler, wie z.B. der Amerikaner SACHS, setzten sich an den Rechner, nahmen ein Malprogramm namens Deluxe Paint zur Hand und begannen Ihr Werk. Das Resultat waren viele sehr schöne Bilder von Landschaften, alltäglichen Gegenständen und Personen. Durch das hohe Auflösungsvermögen und die Farbenvielfalt des AMIGA hatten diese 'Gemälde' eine hohe Realitätsnähe, die durch Schattierungen und Farbabstufungen erreicht wurden. Kein Rechner in der Sparte des AMIGA vermag bisher diese Qualität zu überbieten.
Nachdem der AMIGA einige Zeit auf dem Markt war, tauchten die ersten digitalisierten Bilder auf, deren Qualität jeden Betrachter erstaunte. Besonders 'echte' Bilder wurden im HAM-Modus aufgenommen, da mit den dann verfügbaren 4096 Farben auch sehr feine Farbnuancen wirklichkeitgetreu wiedergegeben werden konnten.
Auf den KICKSTART PUBLIC DOMAIN-Disketten befindet sich eine Auswahl der besten Bilder, die zum einen von Künstlern gemalt und zum anderen von Vorlagen digitalisiert wurden. Im folgenden werden diese Disketten aufgeführt:

Diskette 9: Bilder-Show



Grafik-Show mit bekannten Cartoons und schönen Landschaftsbildern
(IFF-Format)

Diskette 12: Bilder
Digitalisierte Bilder mit erstaunlicher Qualität
(IFF-Format)

Diskette 13: Bilder
Bilder-Show (IFF-Format)

Diskette 19: Bilder-Show
Sehr schöne digitalisierte H.A.M.-Bilder

Diskette 23: Bilder-Show



Viele abwechslungsreiche Motive in verschiedenen Auflösungen, verpackt in einer Grafik-Show (IFF-Format)

Diskette 24: Bilder-Show
Sehr schöne, digitalisierte Frauengesichter

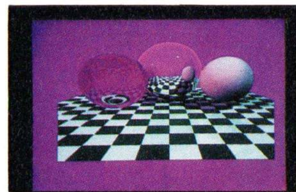
Disketten 26 & 27: Bilder-Show
Auf zwei randvollen Disketten erleben Sie eine einmalige Dia-Show mit hervorragend digitalisierten futuristischen Bildern in voller PAL-Auflösung. Dazu gibt es stimmungsvolle, sphärische Musik. (IFF-Format)

Diskette 39: Bilder-Show
Stimmungsvolle Landschaftsbilder, die sich gut zum Weiterverarbeiten eignen und einige digitalisierte Bilder.
(IFF-Format)

RAYTRACING:

Raytracing ist ein Verfahren zum Berechnen von räumlich wirkenden Bildern. Besonders effektvoll ist dabei der Einsatz von geschickt angeordneten Spiegelfächen, in denen die Figuren immer wieder auftauchen.

Diskette 11: Bilder-Show



RAY-TRACERS: wunderschöne räumliche Bilder, die auf einer VAX berechnet und auf den AMIGA übertragen wurden

Diskette 42: Bilder-Show
Vielfältige, nach dem RAY-TRACER-Verfahren erstellte Bilder. Lassen Sie sich von den realistischen Spiegelungen beeindrucken! Mit digitalisierter Musik!
(IFF-Format)

Diskette 46: Bilder-Show
Eine weitere Diskette (siehe auch PD 42) mit phantastischen RAY-TRACER-Bildern, unterlegt mit digitalisierter, fetziger Musik.
(IFF-Format)

Diskette 60: RAY-TRACER
DBW-Render ist ein leitungsfähiges Programm zum Erstellen von Bildern nach dem Ray-Tracing-Verfahren. Die Daten der Bilder werden mit einem Texteditor eingegeben, wobei spezielle Befehle zur Verfügung stehen. Die Berechnung der Bilder durch das Programm kann, wegen des aufwendigen Verfahrens, mehrere Stunden dauern, aber die Ergebnisse sind hervorragend (siehe Bild).

ANIMATIONEN:

Jeder AMIGA-Besitzer und auch die Nichtbesitzer geraten in Verzückung, wenn dieses 'Grafikwunder' bewegte Vorgänge zeigt. Einige dieser Animationen sind wegen ihrer unübertroffenen Qualität sicherlich bekannt. Besonders der Jongleur (Diskette 10) und die laufende Katze (Diskette 43) sollten in keiner Sammlung fehlen.

Diskette 10: Grafik-Animation



JUGGLER-DEMO: ein bewegliches Männchen jongliert mit drei verspiegelten Kugeln, sehr schöne Demo.

Diskette 15: Grafik-Animation
Verschiedene Filme, die mit dem AEGIS-ANIMATOR erstellt wurden, incl. PLAYER zum Abspielen der Filme. INFO: Einige Filme benötigen auf dem AMIGA 1000 mehr als 512 KB Speicher.

Diskette 20: Grafik-Show
'Fred the Baker und Rose's Flower Shop'
COMIC-Film, der die Multitasking-Fähigkeiten des AMIGA erklärt

Diskette 33: Grafik-Animation
Einige sehr gute, mit Deluxe Video erstellte Filme. Der benötigte PLAYER ist auch auf der Diskette.

Diskette 43: Grafik-Animation



Eine einmalige Show, bei der eine digitalisierte Katze in gleitenden Bewegungen über den Bildschirm trabt. Erstellt wurde diese faszinierende Animation mit einem Digitizer, DPaint und VideoScape 3D.

Diskette 59: Grafik-Animation
Mit drei herrlichen Grafikdemos stellt Eric Graham seine Programme Sculpt 3-D und Animate 3-D vor. Die Bilder der Animationen sind nach dem Ray-Tracer-Verfahren berechnet.

DEMOS:

Diskette 40: GRAFIK-DEMOS
Boing!, Rotate, Sparks, Moire, Dazzle, 3DCube, Scales, Sizzlers.
Sehenswert ist der Film 'Atari meets AMIGA', der die erste und einzige Begegnung der beiden Computer dokumentiert.
Sehr schön ist das Programm LANDSCAPE, das wunderschöne fraktale Berg- und Tallandschaften erzeugt.

Diskette 48: Crazy
Auf dieser Diskette befinden sich nur verrückte Programme, deren Sinn absolut zweifelhaft ist. Allerdings sollten Sie sich diesen Spaß nicht entgehen lassen!
Von fast allen Programmen ist der Source-Code (die Sprache wird in Klammern angegeben!) zusätzlich auf der Diskette.

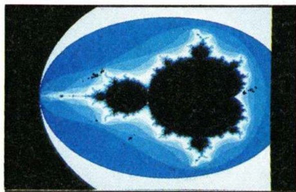
Target : geräuschvolles Mausklicken
RoboTroff : Wer klaut denn da den Cursor? [C]
Smush : Bild x4
Tilt : kippt den Screen [C]
Scat : Was hat denn das Fenster gegen Dich?
Ing : läßt die Fenster hüpfen [C]
DropShadow : jedes Fenster bekommt einen Schatten
DropCloth : endlich bekommt die Werkbank einen Untergrund
MouseClock : nun hat man die Uhrzeit immer im Blickfeld [C]
Dk : winterlich schneit alles herunter [M]
Melt : der Bildschirm zerfließt vor Deinen Augen, von Leo Schwab [C]
Nart : von Leo Schwab [C]
Flip : dreht den Bildschirm auf den Kopf, von Mike Berro[A]

KICKSTART PUBLIC DOMAIN SERVICE

MANDEL-BROTGENE-RATOREN:

Diskette 18: Grafik
Dieser Mandelbrot-Generator (Mandelbrot Set Explorer) ist sehr komfortabel mit der Maus zu bedienen. Er arbeitet in 4 Auflösungsstufen und kann neben anderen Optionen auch dreidimensionale Grafiken erzeugen.

Diskette 38: GRAFIK



NoFFP Mandelbrot Set Explorer V2.1 (neue Version) von ABC Softarts in Braunschweig
Dieser Generator arbeitet mit PopUp-Menus, eigenen Floatingpoint-Routinen zur Beschleunigung der Berechnungen und nutzt den gesamten PAL-Bildschirm.

UTILITIES:

Diskette 41: UTILITIES (Grafik)
Alles, was Sie zu dem von ELECTRONIC ARTS entwickelten Grafik-Standard (IFF-Format) wissen müssen: Laden, Speichern, Komprimieren, Dekomprimieren. Mit Dokumentationen und Source-Codes in C.

Diskette 49: ICONS
Utility-Programme, die sich mit der Erstellung und Manipulation von Icons beschäftigen.

XICON 2.0 : Mit diesem Programm können Dateien ausgeführt werden, die CLI-Kommandos enthalten. Fast jedes Programm, das sich normalerweise nur über das CLI starten läßt, kann nun auch über ein Icon angeklickt werden. Das Programm wurde des öfteren verbessert und liegt hier als ausgereiftes und benutzerfreundliches Produkt vor.
Autor: Pete Goodeve
[ICON][DOC]

IconMk 1.2a : Dieses Programm versieht Dateien mit einem ICON, die davor keines hatten.
[ICON][DOC]

Icon2C : wandelt ein Icon in eine C-Struktur um

ImageTools : ist eine Ansammlung von Hilfsprogrammen, die sich mit der Erstellung und Manipulation von Icons beschäftigen. Damit können die Größe von IFF-Bildern und die Anzahl der Farben reduziert werden, ohne das Bild zu stark zu verändern.

Diskette 55: Grafik/Utilities
Einige schöne Grafikdemos und Utilities zu diesem Thema

ShowPrint: ein interessantes Utility für alle Grafikfans. Alle IFF-Bildformate können geladen und ausgedruckt werden.

WBLander: ein kleines Spielchen, bei dem auf der WB eine Rakete erscheint, die man auf einem Fenster landen muß. Diese Version ist mit digitalisiertem Sound und neuen Grafiken ausgestattet.

SHM: eine interessante Grafikdemo mit Source-Code

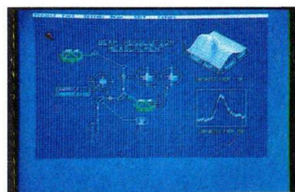
LineDrawer: Dieses Programm zeichnet beliebige Linien, deren Daten es einem Datenfile entnimmt. Die Größe des Bildes ist dabei allein von der des Fensters abhängig. Als Beispiel wird eine Karte der USA gezeichnet.

Nemesis: Diese Grafikdemo erreichte den 5. Platz beim 'Badge Killer Demo Contest'

DEMOlition: hinter diesem Namen verbirgt sich Unfug

CAD:

Diskette 36: CAD



mCAD ist ein wirklich gut gemachtes CAD-Programm, das jedoch nur im Interlace-Modus läuft. Es bietet die einfachen Zeichenfunktionen und Features wie Zoom, Group, Ungroup, Grid, Move, Rotate. Auf der Diskette sind mehrere Dokumente, die das Programm erklären und einige Zeichnungen, die mit mCAD erstellt wurden.

WICHTIG:

Die Programme laufen auf allen AMIGA-Computern mit Kickstart/Workbench 1.2, allerdings sollten 512k Speicher vorhanden sein, einige Grafikanwendungen verlangen sogar mindestens 1MB Speicher. Sollten dennoch Einschränkungen gelten, wird dies bei den betreffenden Programmen angegeben.

Versandbedingungen:
Um einen schnellen und problemlosen Versand zu gewährleisten, beachten Sie bitte folgende Punkte:

- Für jede Diskette ergibt sich ein Unkostenbeitrag von DM 10.-
- Pro Sendung kommt ein Versandkostenbetrag (für Porto und Verpackung) von DM 5.- (Ausland DM 10.-) hinzu. Ins Ausland können wir nur gegen Vorrauskasse liefern.
- Bitte legen Sie außerdem einen Aufkleber mit Ihrer Adresse bei

Anschrift:
MERLIN Computer GmbH
KICKSTART Redaktion
Postfach 55 69
6236 Eschborn

Die Diskettenbestellung kann auch telefonisch erfolgen. Der Versand erfolgt dann per Nachnahme.

Tel.: 06196/ 48 18 11
(MO.-FR. von 9-17 Uhr)



VON CHRISTIAN SCHORMANN

RAY TRACING

Verfolgungsjagd mit dem Computer

Computergrafiker versuchen, immer realistischere Bilder mit dem Computer zu erzeugen. Die Algorithmen dafür werden ständig weiterentwickelt. Bisher hat es die mangelnde Rechenleistung, Auflösung und Farbauflösung von Mikrocomputern unmöglich gemacht, solche Algorithmen auch auf Mikrocomputern zu implementieren.

Seit kurzem aber gibt es Programme wie Sculpt 3D oder Silver, die Grafiken und Animationen von erstaunlicher Qualität auf dem Amiga erzeugen. Wie das dabei verwendete Verfahren funktioniert, erklärt dieser Artikel..

Das Ray Tracing-Verfahren wurde ursprünglich als Hidden Surface-Verfahren entwickelt. Ein Hidden Surface-Verfahren dient dazu, festzustellen, welche Teile eines 3D-Objektes vom Standpunkt des Betrachters aus sichtbar sind (und deshalb gezeichnet werden müssen) und welche nicht.

Schon sehr schnell stellte sich aber heraus, daß das Verfahren viel mehr Möglichkeiten bietet: Es ist nämlich ohne weiteres möglich, mit dem gleichen, einfach zu implementierenden Algorithmus gleichzeitig auch Schatten, transparente Körper und Lichtspiegelungen jeder Art zu erzeugen.

Die Grundidee ist einfach. Ein Betrachter einer beliebigen Szene sieht Licht, das von irgendwelchen Lichtquellen (Lampen, Sonnen, Feuer, Kerzen usw.) ausgesandt und von den Gegenständen, aus denen die Szene besteht, in Richtung des Betrachters reflektiert wird. Die Farbe eines Gegenstandes hängt dabei von seinen Reflektionseigenschaften ab. Licht setzt sich aus Wellen verschiedener Wellenlänge zusammen, wobei uns langwelliges Licht rot, kürzerwelliges Licht blau erscheint. Dazwischen liegen die Farben des Regenbogens. Ein roter Körper reflektiert im wesentlichen rotes Licht, andere Lichtwellen werden absorbiert. Ein ideal schwarzer Körper reflektiert überhaupt kein Licht. Etwas so egoistisches gibt es aber in der Realität nicht. Weißes Licht ist eine Mischung aus allen Lichtfarben, was leicht erkennbar ist, wenn man Sonnenlicht durch ein Prisma zerlegt. Ein Prisma lenkt Licht entsprechend seiner Wellenlänge mehr oder weniger ab; doch damit erst einmal genug über Licht. Um festzustellen, was der Betrachter sieht, verfolgt man einfach jeden Lichtstrahl von seinem Entstehungsort an und beobachtet, wo er hinstrahlt, wohin er reflektiert wird, usw. Oder nein, vielleicht ist das keine allzugute Idee, denn von den Lichtstrahlen, die eine Lichtquelle aussendet, erreicht nur ein verschwindend kleiner Teil einen Betrachter. Außerdem sendet so eine Lichtquelle unendlich viele Lichtstrahlen aus, was zu einer etwas unpraktisch langen Rechenzeit führen könnte. So geht es also nicht. Aber umgekehrt... Wenn man einfach vom Auge des Betrachters aus alle Lichtstrahlen in die Szene verfolgt, hat man schon mal entschieden viel weniger zu tun. Aber immer noch unendlich viel. Man muß sich also noch etwas einfallen lassen, das die Anzahl der zu verfolgenden Strahlen begrenzt. Dazu fällt uns sofort der Bildschirm ein. Schließlich müssen wir das Bild sowieso auf einem endlichen Raster, z.B. auf 640*400 Punkten darstellen. Also verfolgen wir nur diejenigen Strahlen, die vom Betrachterstandpunkt (wir nehmen selbstverständlich einen punktförmigen Betrachter an - trinken Sie mehr Bier, dann runden Sie sich von selbst mit der Zeit) aus durch ein Pixel des Bildschirms auf die Szene fallen. Um die Rechnerei zu vereinfachen, nehmen wir einfach mal an, der Bildschirm läge auf der xy-Ebene. In Bild 1 sehen Sie, wie das aussieht. Jeder Strahl muß sich irgendwie mathematisch beschreiben lassen, ohne Steckbrief können wir ihn schlecht verfolgen. Zwei Punkte des Strahls sind uns bekannt, die Koordinaten des Pixels, dessen Strahl wir gerade verfolgen und die Koordinaten des Blickpunktes. Daraus läßt sich glücklicherweise eine Geradengleichung konstruieren, und zwar derjenigen Geraden, die durch das Pixel und den Blickpunkt geht.

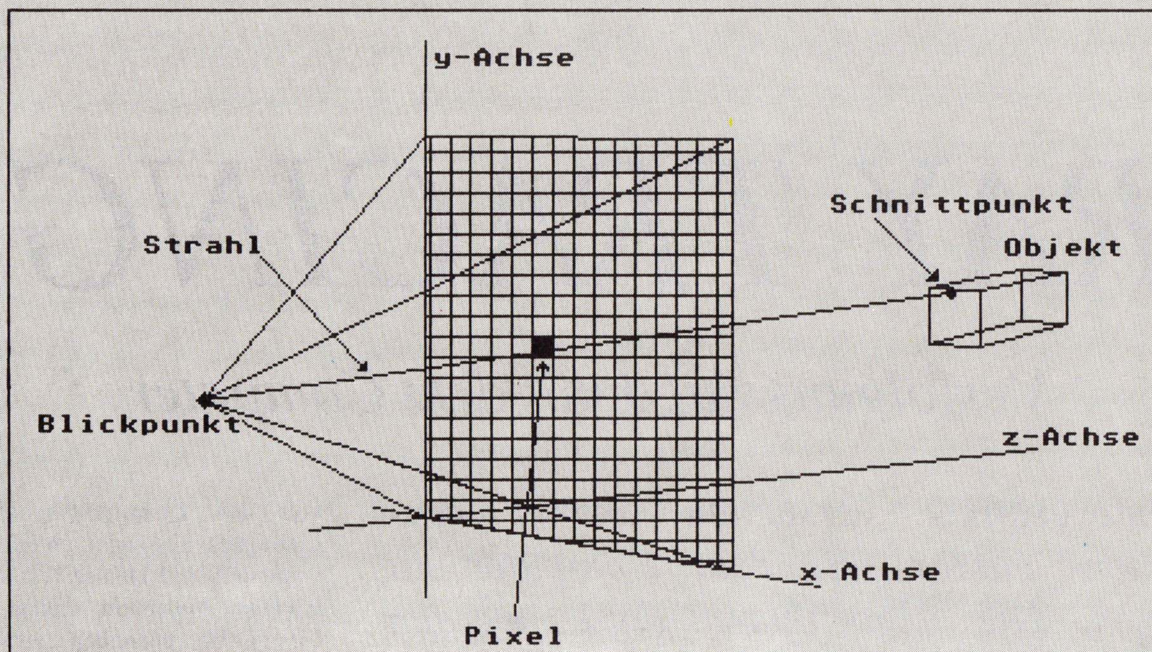


Bild 1: Ray Tracing
Ein Strahl durch den Blickpunkt und eines der Pixel der Bildebene wird auf Schnittpunkte mit den Objekten der Szene untersucht. Bei mehreren Schnittpunkten wird der der Bildebene am nächsten liegende Schnittpunkt weiterverwendet.

Es gilt:

$$x = x1 + (x2 - x1)*t,$$
$$y = y1 + (y2 - y1)*t,$$
$$z = z1 + (z2 - z1)*t,$$

wobei x,y,z ein beliebiger Punkt auf dem Strahl ist, x1,y1,z1 der Blickpunkt und x2,y2,z2 das entsprechende Bildschirmpixel. Da wir die xy-Ebene für den Bildschirm gewählt haben, ist z2 natürlich gleich 0. Wenn t=0 ist, erhält man als Ergebnis der Gleichung den Blickpunkt, wenn t=1 ist, ergibt sich das Pixel. Daraus kann man entnehmen, daß t>1 wird, wenn wir den Strahl weiter in die Szene verfolgen. Warum tun wir das überhaupt ? Nun, wir wollen wissen, welches Objekt vom Blickpunkt aus, wenn man sozusagen entlang des Strahls durch das Pixel schaut, sichtbar ist. Dieses Pixel müssen wir dann nämlich entsprechend dem gefundenen Objekt verändern. Finden wir ein blaues Objekt, muß dieses Pixel blau werden, ist es rot, wird das Pixel eben auch rot. Um festzustellen, auf welches Objekt der Strahl trifft, berechnen wir einfach Schnittpunkte zwischen dem Strahl und allen Objekten der Szene. Wenn der Strahl kein Objekt trifft, verläßt er die Szene, und das Pixel erhält die Farbe des Hintergrundes (z.B. des blauen Himmels über dem Kornfeld). Trifft er nur ein Objekt, nun, dann ist alles klar. Gibt es aber Schnittpunkte mit mehreren Objekten, müssen wir wissen, welcher Schnittpunkt dem Beobachter am nächsten liegt (denn dieses Objekt verdeckt dann alle anderen). Das ist allerdings recht einfach. Am Pixel ist der Wert für t gleich 1, am Blickpunkt 0. Wenn sich jetzt für die Schnittpunkte Werte für t zwischen 2 und 7 ergeben, liegt natürlich der Punkt mit dem kleinsten Wert für t dem Bildschirm (und damit auch dem Beobachter) am nächsten.

Wenn alle Schnittpunkte aller Strahlen (sprich eines Strahles durch jedes Pixel) mit jeweils jedem Objekt berechnet worden sind, ist der Algorithmus am Ende (Ihre Geduld wahrscheinlich auch). Für jedes Pixel wurde jetzt festgestellt, welches Objekt von dort aus sichtbar ist. Dies ist die einfachste Aufgabe des Ray Tracing-Verfahrens, nämlich das Erkennen aller sichtbaren Objektteile. Vielleicht ist Ihnen aufgefallen, daß dies im Prinzip eine Digitalisierung der dreidimensionalen Welt ist. Eigentlich treffen unendlich viele Strahlen den Beobachter, sie wählen jedoch nur eine endliche Anzahl davon aus. Nichts anderes geschieht beim Digitalisieren von Musik, z.B. für die CD-Produktion; das kontinuierliche Musiksignal wird in einzelne Abtastwerte zerlegt. Wie für Musik ist auch hier das Abtasttheorem von Shannon gültig, nach dem sich immer dann Verzerrungen ergeben, wenn die Abtastrate niedriger als das Doppelte der höchsten im abzutastenden Signal vorkommenden Frequenz ist. Leider sind diese Verzerrungen, die sich bei Rasterbildschirmen als häßliche Treppchen in schrägen Konturen zeigen, nur mit aufwendigen, Anti-Aliasing genannten Verfahren zu mildern. Aber dies nur nebenbei.

Noch etwas anderes. Wie berechnet man Schnittpunkte eines Strahles mit einem Objekt ? Ganz einfach, man benötigt dafür nur eine mathematische Beschreibung des Objektes. Eine Kugel z.B. ist mathematisch sehr einfach zu beschreiben, weshalb beispielsweise das Männchen bei der Juggler-Demo, die wohl jeder Amiga-Besitzer kennt oder sein eigen nennt, aus Kugeln zusammengesetzt ist. Es ist auch möglich, frei geformte Oberflächen mathematisch zu beschreiben, aber dann steigt die ohnehin schon lange Rechenzeit eines Ray Tracers stark an,weil Schnittpunkte mit solchen komplexen Objekten sehr schwer zu bestimmen sind. 640*400 Pixel sind insgesamt 256.000 Pixel. Wenn eine Szene aus zehn Objekten, was wahrlich nicht viel ist, besteht, müssen 256.000 Strahlen mit jeweils 10 Objekten geschnitten werden. Das macht 2.560.000 Schnittpunktberechnungen. Und das sind nur die Schnittpunkte, die für die Hidden Surface-Berechnung gebraucht werden, alle Effekte, die erst die realistischen Bilder ergeben, haben wir ja noch gar nicht erwähnt.

In Bild 2 sehen Sie die Berechnung des Schnittpunktes eines Strahles mit einer Kugel. Mit Zylindern ist es auch nicht viel schwieriger. Die Grundidee ist immer die gleiche; man setzt die Gleichung des Strahles in die des Objektes ein und schaut nach, ob etwas sinnvolles herauskommt. Wenn nicht, gibt es keinen Schnittpunkt.

```
Bild 2:
P1  P2  Ps  Pk  C  r

Jeder Punkt Ps auf einem
Strahl durch die Punkte P1
und P2 erfüllt die Gleichung:
        Ps=P1+Pr*t    mit Pr=(P2-P1)
Jeder Punkt Pk auf der Kugeloberfläche
erfüllt die Gleichung:
        (Pk-C)*(Pk-C)-r²=0
Für einen Schnittpunkt gilt: Ps=Pk.
Jetzt kann man die beiden Gleichungen
ineinander einsetzen und nach t auf-
lösen:
        ((P1+Pr*t)-C)*((P1+Pr*t)-C)-r =0
Nach einigen Umformungen erhält man
eine quadratische Gleichung der Form:
        a*t²+b*t+c=0  mit
a=Pr*Pr
b=2*Pr*(P1-C)
c=(P1-C)²-r²
Wie für jede quadratische Gleichung
gibt es auch hier zwei Lösungen:

   t1,2=(-b±√(b²-4*a*c))/(2*a)

Der Term unter der Wurzel gibt die
möglichen Lösungen an; ist er kleiner
0 gibt es keine Lösung, also auch keinen
Schnittpunkt. Ist er gleich null, berührt
der Strahl die Kugel: 1 Schnittpunkt. Ist
er größer 0 gibt es zwei Schnittpunkte.
```

Special Effects...

Bisher haben wir einfach aufgehört, wenn wir ein erstes Objekt gefunden haben. So geht das aber nicht; das Licht kommt ja nicht von diesem Objekt, sondern von irgendeiner Lichtquelle und ist nur von diesem Objekt reflektiert worden. Die geometrische Optik liefert alle Gesetze, die man zur Berechnung der Lichtbrechung auf einer Oberfläche braucht. Deshalb müssen wir nun feststellen, unter welchem Winkel ein Lichtstrahl die Oberfläche erreicht. Damit kann man dann auch feststellen, in welcher Richtung er die Oberfläche wieder verläßt. Wir tun dies natürlich andersherum; wir wissen ja, wie der Lichtstrahl gekommen sein müßte.

Wir verfolgen also den Weg des Lichtstrahls über alle Ecken und Kanten, bis er die Szene verläßt oder auf eine Lichtquelle trifft. Damit wissen wir, welche Intensität der Lichtstrahl gehabt hat, als er losgesendet wurde. Wenn er von der Lampe kommt, hat er die Intensität der Lampe, kommt er aus dem Hintergrund, die der Umgebungshelligkeit. Bei jeder Reflektion wird diese Intensität eben durch die Reflektion gemindert, wobei die Stärke der Minderung von den Oberflächeneigenschaften des reflektierenden Objektes abhängt. Ein Spiegel mindert wenig, eine matte Oberfläche stark. Wenn man transparente Oberflächen zuläßt, taucht ein weiteres Problem auf. An einer transparenten Oberfläche wird ein Strahl nicht nur gespiegelt, sondern auch gebrochen (Bild 3). Auch hierbei gelten wieder die Gesetze der

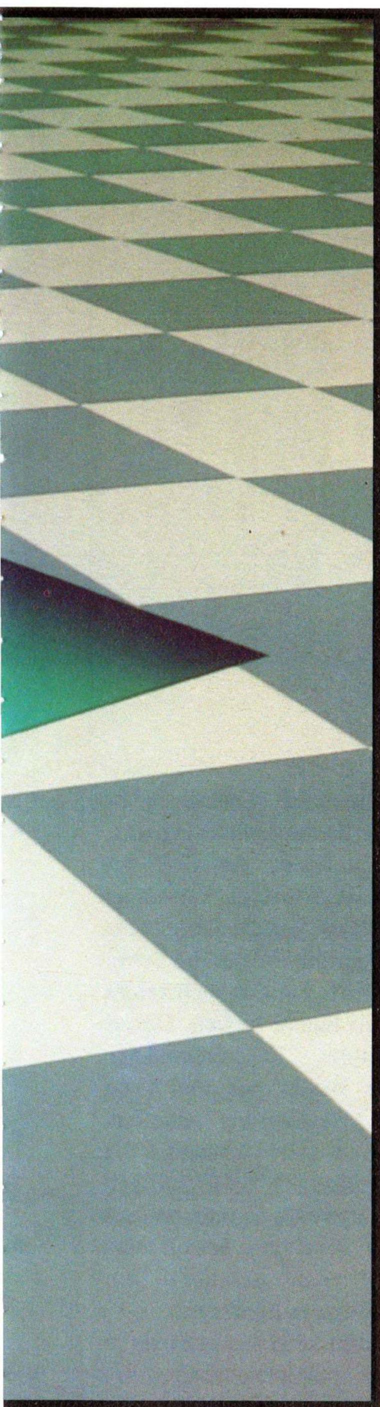


geometrischen Optik. Der erste Schnittpunkt ist im allgemeinen also nicht das Ende der Strahlverfolgung, im Gegenteil, hier spaltet sich der ursprüngliche Strahl in einen Baum von reflektierten und gebrochenen Strahlen auf, die alle verfolgt werden müssen, bis sie die Szene verlassen.

Erst an diesem Punkt, wenn die Ausgangspunkte aller Strahlen, die aus dem Ursprungsstrahl entstanden sind, berechnet wurden, kann damit begonnen werden, die tatsächlichen Helligkeitsverhältnisse am ersten Schnittpunkt zu berechnen. Die Helligkeit an dieser Stelle ist nämlich die Summe aller auf den Punkt auftreffenden Teilhelligkeiten. An jeder Brechungs- oder Reflektionsstelle müssen, vom Ursprungsort ausgehend, die Veränderungen, die die Objekte je nach Materialeigenschaften dem Lichtstrahl angetan haben, berechnet werden. Die Summe all dieser Teilintensitäten ergibt dann die endgültige Helligkeit des Pixels. Es ist leicht ersichtlich, daß die Qualität des Bildes auf der Richtigkeit der Berechnung der Lichtstrahlen, ihrer Helligkeitsminderungen usw. beruht. Die heute verwendeten Lichtmodelle (Bild 3) sind aber so gut, daß sogar das Streulicht an einem bestimmten Punkt berücksichtigt wird, so daß Schlaglichter an spiegelnden Objekten entstehen. Wichtig bei allen Lichtmodellen ist, daß man sie für alle drei Grundfarben der additiven Farbmischung, nämlich rot, grün und blau getrennt berechnen muß. Schließlich hat ja jedes Objekt seine eigene Farbe, d.h. farbabhängige Brechungs- und Reflektionseigenschaften. Wie gesagt, eine rote Kugel ist rot, weil sie rotes Licht besser reflektiert als die anderen Lichtanteile. Außerdem muß natürlich

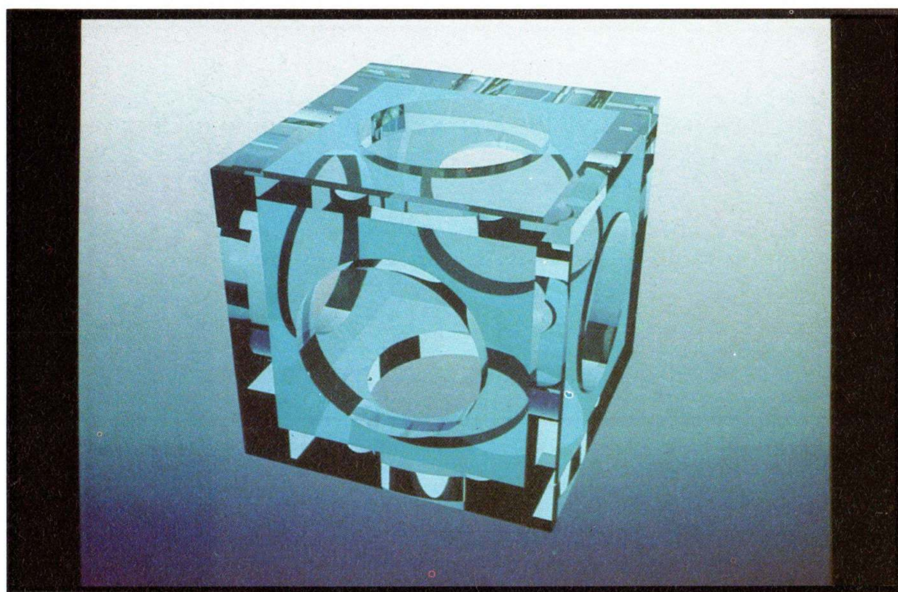


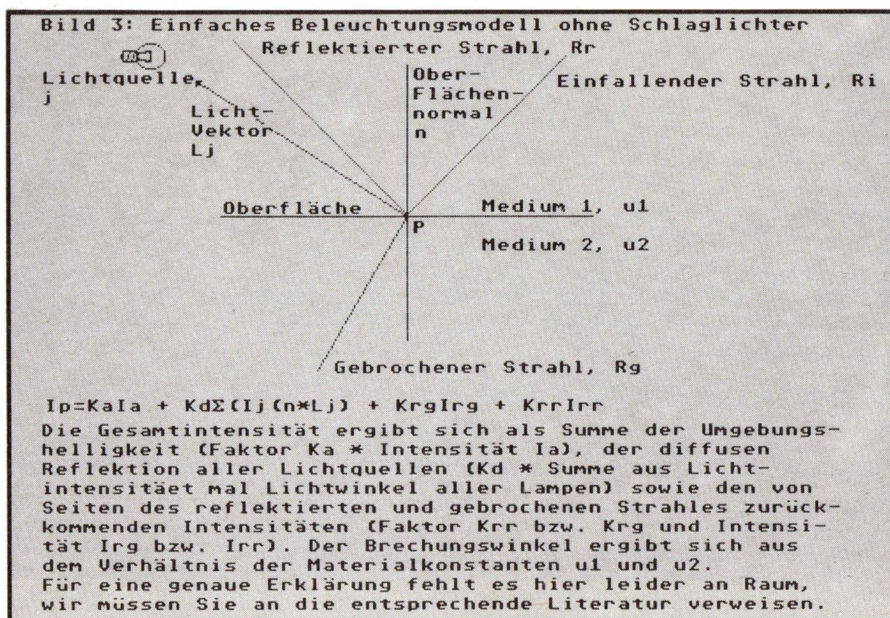
jede Lichtquelle berücksichtigt werden.
Jetzt fehlen nur noch die Schatten, die über unseren Häuptern schweben... Schatten sind aber überhaupt kein Problem. An jedem Schnittpunkt konstruiert man außer den gebrochenen und reflektierten Strahlen sogenannte "Shadow Feeler", zu gut deutsch Schattenfühler. Das sind Strahlen, die vom Schnittpunkt aus in Richtung auf jede vorhandene Lichtquelle zeigen. Diese Strahlen müssen jetzt wiederum mit jedem Objekt geschnitten werden. Liegt kein Objekt zwischen Schnittpunkt und Lichtquelle, ist der Schnittpunkt von dieser Lichtquelle voll beleuchtet. Liegt ein nichttransparentes Objekt dazwischen, liegt der Schnittpunkt auf diese Lichtquelle bezogen im Schatten. Ist das Objekt dazwischen transparent, entsteht ein Halbschatten, die Wirkung der Lichtquelle auf den Schnittpunkt muß gemindert werden (Bild 4).
Sie sehen, die Anzahl der Schnittpunktberechnungen hat sich inzwischen vervielfacht. Für jede Reflektion und jede Brechung müssen wieder Schnittpunkte der entstehenden Strahlen berechnet werden, für jeden Schnittpunkt müssen vor Anwendung des Lichtmodells Schattenfühler für jede Lichtquelle berechnet und auch diese mit jedem Objekt geschnitten werden. Bei Szenen mit vielen Objekten kann das leicht zu 100 (oder auch erheblich mehr) Schnittpunktberechnungen pro Pixel führen. Und diese sind bei komplexen Objekten auch erheblich komplizierter als bei Kugeln. Kein Wunder also, daß man auch nach Einsatz aller Beschleunigungstechniken immer noch davon ausgeht, daß 800-1000 MFlops für die Real-Time-Berechnung von Ray Tracing-Bildern benötigt werden. Selbst die schnellsten heute verfügbaren Computer benötigen mindestens 10 Minuten für ein Bild, an dem ein Amiga Jahre zu rechnen hätte. Dennoch, der Aufwand lohnt sich. Keine andere Technik kann so realitätsnahe Computerbilder erzeugen wie eben Ray Tracing. Und bei wirklich komplexen Szenen ist es auch um die Effizienz der anderen Techniken, die von den Objekten und nicht dem Bildschirm ausgehen, nicht mehr so gut bestellt.



Verbesserungen

Das Wort "Beschleunigungstechniken" im vorigen Absatz hat Sie hoffentlich neugierig gemacht. Es wurden, gerade in neuester Zeit, einige Algorithmen entwickelt, die Ray Tracing erheblich effizienter machen, und vor allem den fatalen Effekt, daß die Rechenzeit mit steigender Objektanzahl exponential ansteigt (warum wohl?), auf mindestens lineare Verhältnisse mildern. Eine erste Maßnahme ist es, die Umrisse der darzustellenden Objekte auf konventionelle Weise rechnerisch auf die Bildebene zu projezieren. Dann müssen nur noch die Strahlen verfolgt werden, die innerhalb der Umrisse liegen. Bei sehr komplizierten oder sehr vollen Szenen nützt das leider praktisch nichts.
Sehr gebräuchlich ist die Zusammenfassung mehrerer beieinander liegender Objekte in einer Hierarchie. Beispiel: Objekte sind ein Tisch, auf dem eine Obstschale mit Äpfeln steht. Jetzt kann man diese ganze Gruppe mit einem Volumen umschließen, einer Kugel oder Box etwa. Ein Strahl, der dieses Volumen nicht schneidet, kann auch unmöglich eines der Objekte im Inneren schneiden.





Die Obstschale wird ebenfalls samt Inhalt in ein Testvolumen gepackt. Wenn nun ein Strahl die Box der gesamten Szene schneidet, kann man wiederum die Box um die Obstschale testen. Auch hier gilt: Wenn der Strahl die Box nicht trifft, kann er auch keines der Objekte im Innern treffen. Dieses Verfahren nutzt Zusammenhänge innerhalb des Bildes aus und kann dann einige Erleichterung schaffen. Problematisch ist unter Umständen die Auswahl eines vernünftigen Umhüllungsvolumens: es muß einfach zu berechnen sein (am besten eine Kugel), aber auch möglichst gut passen, sonst nützt es nicht viel.

Es gibt ein Verfahren, das erstmals in einem Artikel vom letzten Jahr ('87) beschrieben wurde. Dieses nutzt den



Umstand aus, daß entsprechend dem Verhalten natürlicher Objekte jedes Pixel den umliegenden Pixeln verhältnismäßig ähnlich ist. Beispiel: Wenn Sie eine Fläche bearbeiten, werden alle Pixel, außer den am Rand liegenden, nur leichte Änderungen gegenüber den umliegenden Pixeln aufweisen. Die Gemeinsamkeiten zwischen den Bildteilen eines Objektes lassen uns dieses Objekt ja erst als solches erkennen. Diese Eigenschaft kann man zu einer Vorsortierung aller möglichen Strahlen nutzen, womit sich ein annähernd konstantes Verhalten des Algorithmus erreichen läßt. Einer der bestechenden Vorteile des einfachen Ray Tracers, seine simple Implementierung, geht damit aber verloren.

Eine ganze Reihe von Verbesserungen basieren auf einem anderen Prinzip. Hier wird der an sich kontinuierliche, "analoge" Raum in einzelne Raumzellen zerlegt, ähnlich wie ein Rasterbildschirm die Fläche des Bildschirms durch ein Raster aus Pixeln darstellt. Hier gibt es zwei Ansätze; der meines Erachtens interessanteste ist ganz neu und basiert auf einer Zerlegung des Raums in gleichmäßig große Raumzellen, die exakt den Pixeln des Bildschirms entsprechen. Zum Verfolgen eines Strahles wird jetzt auch keine mathematische Berechnung verwendet, sondern ein Verfahren, das eine dreidimensionale Erweiterung eines Algorithmus zum Linienziehen im Zweidimensionalen darstellt. Dieses Verfahren hat den Vorteil, daß ausschließlich Integer-Arithmetik und nicht eine einzige Division verwendet werden muß. Der Strahl wird einfach als Linie im 3D-Raum in die Raumzellen hineingezogen. Wenn er auf eine nichtleere Zelle stößt, ist dies automatisch der Schnittpunkt mit dem nächsten Objekt, der Strahl pflanzt sich ja vom Bildschirm aus beginnend in seiner Richtung fort. Damit spart man sich langwierige Schnittpunktberechnungen mit Objekten, die am Ende den Strahl gar nicht berühren. Die Geschwindigkeitsgewinne des Verfahrens sind enorm: Bilder, die auf traditionelle Weise auf einer VAX vorsichtig geschätzt 40 Tage Rechenzeit benötigt hätten, waren nach dem neuen Algorithmus in zweieinhalb Stunden fertig. Der andere Ansatz zerlegt den Raum in eine Hierarchie von verschieden großen Zellen. Leere Raumgebiete werden zu großen, leeren Zellen, andere Gebiete werden solange unterteilt, bis die Zellen klein genug sind, um eine einfache Beschreibung des Objektteiles zu ermöglichen. Die Strahlverfolgung kann auch in dieser Variante mit einem 3D-Linienalgorithmus erfolgen, es geht aber auch anders. Die Resultate dieses Verfahrens sind nicht ganz so günstig wie die des vorigen und zudem von der Kohärenz der Szene abhängig. Auch für bestimmte, hochkomplexe Objekttypen wie frei formbare Oberflächen oder fraktale Landschaften gibt es Verfahren, die die bei diesen Objektformen extrem aufwendigen Schnittpunktberechnungen stark beschleunigen. Eine letzte Gruppe von Verbesserungen befaßt sich mit der Bildqualität. Auch mit Ray Tracing ist es in seiner Grundform nicht möglich, das gesamte physikalische Verhalten von Licht zu simulieren. Neuere Ergänzungen setzen sich zum Ziel, auch die Lichtbeugung an einem Spalt oder die Diffusion in nebelartigen Atmosphären berechnen zu können. Aber dies geht weit über diesen Artikel hinaus. Bei all diesen Veränderungen und Verbesserungen sieht man den ursprünglich so einfach zu implementierenden Algorithmus mit Tränen in den Augen zu einem hochkomplexen Monstrum werden...



Implementierungs-hinweise

Mit ein paar Hinweisen zur Implementierung soll dieser Artikel seinen Abschluß finden.

Es bietet sich förmlich an, den Algorithmus so aufzubauen, daß Strahlen auf einen Stack abgelegt werden. Vor dem Aufruf der eigentlichen Tracer-Routine berechnet ein init-Teil den Ausgangsray für ein bestimmtes Pixel, schiebt ihn auf einen Stack und ruft dann den Tracer auf.

Dieser holt einen Strahl vom Stack und berechnet Schnittpunkte. Wenn er den nächsten gefunden hat, berechnet er die entstehenden Strahlen und speichert auch diese auf dem Stack. Dann beginnt er von vorne und wiederholt diesen Vorgang solange, bis alle entstehenden Strahlen ins Leere gelangen. Jetzt geht es umgekehrt: Der Reihe nach werden die Strahlen vom Stack geholt; für ihre Schnittpunkte, die natürlich in einer Datenstruktur "Ray" gespeichert sein müssen, wird das Beleuchtungsmodell angewendet. Das Beleuchtungsmodell sollte die Schattenfühler berechnen und damit sein Ergebnis beeinflußen, das zur Helligkeit des jeweiligen Pixels addiert wird.

Wenn der Stack leer ist, ist man wieder am Ausgangspunkt angelangt und kann das Pixel auf die berechnete Helligkeit setzen.

Literatur

Eine sehr gute und ausführliche Einführung in Ray Tracing und Beleuchtungsmodelle, die allerdings umfassende Englisch- und Mathematikkenntnisse voraussetzt, findet sich in
David F. Rogers
"Procedural Elements for Computer Graphics",
erschienen im McGraw-Hill-Verlag.

Beschleunigungsverfahren werden in folgenden Publikationen erläutert:

Fujimoto/Tanaka/Iwata
ARTS: Accelerated Ray Tracing System
IEEE CG&A, April 1986, pp.16-26

Kunii (Ed.)
Computer Graphics 1087
Springer Verlag, Berlin

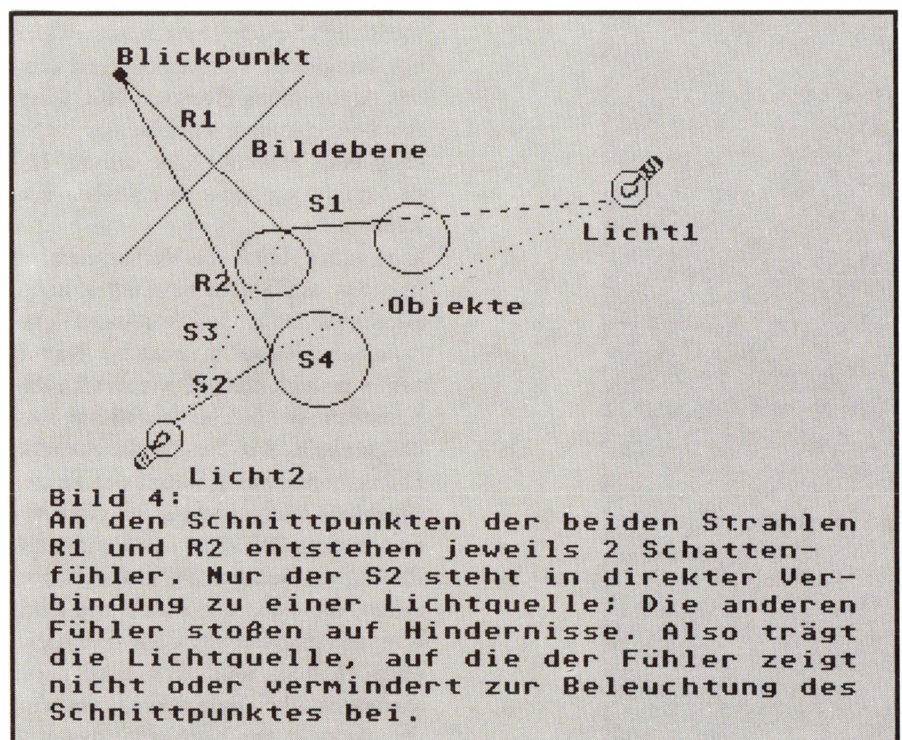


Bild 4:
An den Schnittpunkten der beiden Strahlen R1 und R2 entstehen jeweils 2 Schattenfühler. Nur der S2 steht in direkter Verbindung zu einer Lichtquelle; Die anderen Fühler stoßen auf Hindernisse. Also trägt die Lichtquelle, auf die der Fühler zeigt nicht oder vermindert zur Beleuchtung des Schnittpunktes bei.

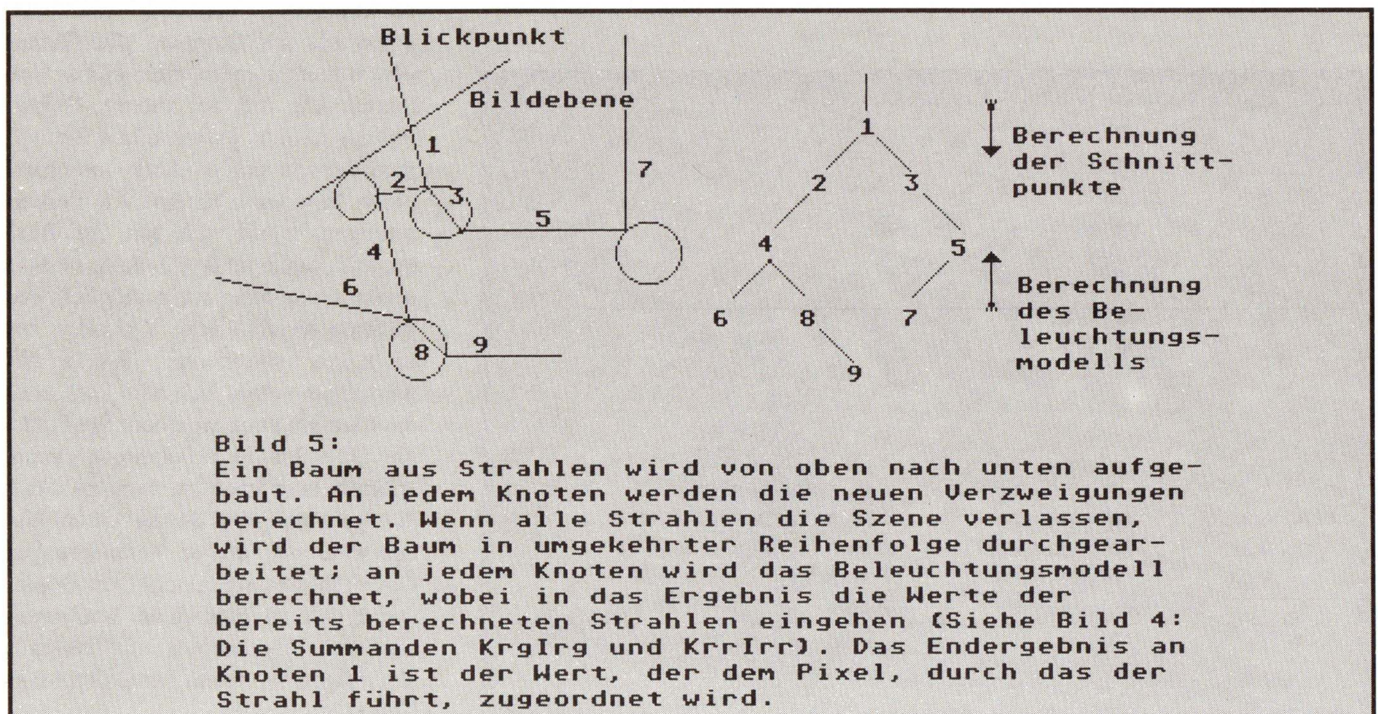


Bild 5:
Ein Baum aus Strahlen wird von oben nach unten aufgebaut. An jedem Knoten werden die neuen Verzweigungen berechnet. Wenn alle Strahlen die Szene verlassen, wird der Baum in umgekehrter Reihenfolge durchgearbeitet; an jedem Knoten wird das Beleuchtungsmodell berechnet, wobei in das Ergebnis die Werte der bereits berechneten Strahlen eingehen (Siehe Bild 4: Die Summanden KrgIrg und KrrIrr). Das Endergebnis an Knoten 1 ist der Wert, der dem Pixel, durch das der Strahl führt, zugeordnet wird.


KiCK
START


KiCK


KLARE ANSICHTEN

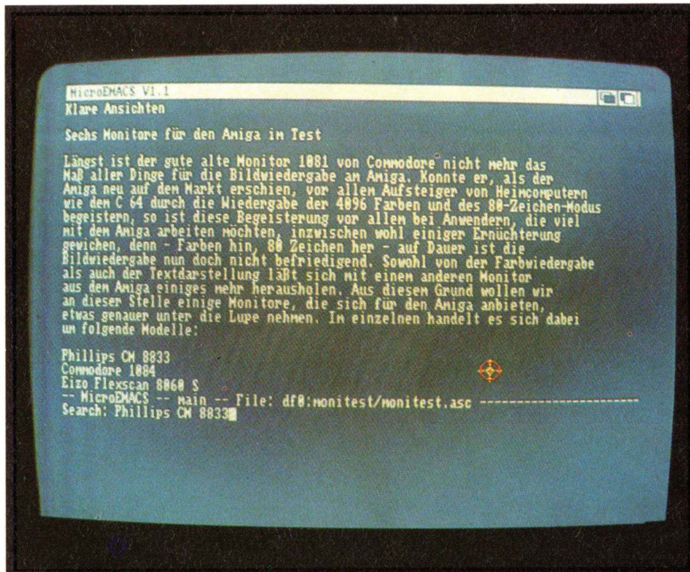
VON WOLF DIETRICH



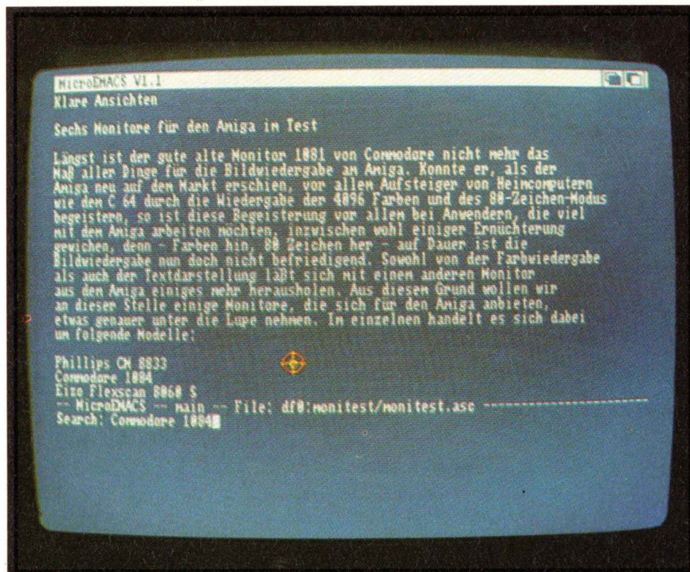
SECHS MONITORE FÜR DEN AMIGA IM TEST

Längst ist der gute alte Monitor 1081 von Commodore nicht mehr das Maß aller Dinge für die Bildwiedergabe am Amiga. Konnte er, als der Amiga neu auf dem Markt erschien, vor allem Aufsteiger von Heimcomputern wie dem C 64 durch die Wiedergabe der 4096 Farben und des 80-Zeichen-Modus begeistern, so ist diese Begeisterung vor allem bei Anwendern, die viel mit dem Amiga arbeiten möchten, inzwischen wohl einiger Ernüchterung gewichen, denn - Farben hin, 80 Zeichen her - auf Dauer ist die Bildwiedergabe nun doch nicht befriedigend.

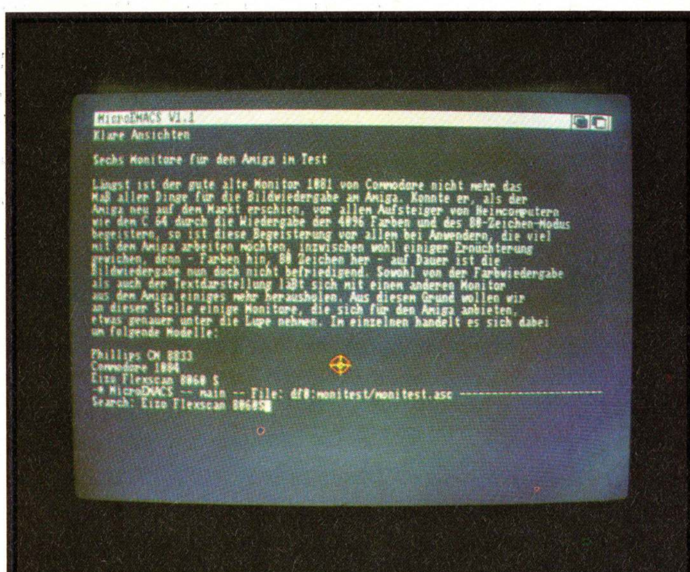
Sowohl von der Farbwiedergabe als auch der Textdarstellung läßt sich mit einem anderen Monitor aus dem Amiga einiges mehr herausholen. Aus diesem Grund wollen wir an dieser Stelle einige Monitore, die sich für den Amiga anbieten, etwas genauer unter die Lupe nehmen. Im einzelnen handelt es sich dabei um folgende Modelle:



Textwiedergabe auf dem Phillips CM 8833



Textwiedergabe auf dem Commodore 1084



Textwiedergabe auf dem Eizo 8060 S

Phillips CM 8833
Commodore 1084
Eizo Flexscan 8060 S
Santec DMC 1537
Mitsubishi HF 1400
NEC Multisync

Der Commodore 1081 ist in diesen Test nicht mit einbezogen, dient aber bisweilen als Maßstab.

Bei den Monitoren von Commodore und Phillips handelt es sich um Modelle, die von den Anschlußwerten bzw. der vertikalen und horizontalen Bildwiederholungsfrequenz exakt den Anforderungen des Amiga entsprechen. Beide sind außerdem in der Lage, das CGA-Signal eines PC oder Kompatiblen zu verarbeiten. Die anderen getesten Modelle sind in der Lage, sich automatisch auf verschiedene Frequenzen einzustellen; es handelt sich bei ihnen um sogenannte Multiscan-Monitore. Sie verarbeiten somit nicht nur Signale eines Amiga, sondern beispielsweise auch Signale einer PC-EGA-Grafikkarte, was unter Umständen für Amiga 2000-Anwender, die auch eine PC-Karte und eine EGA-Grafikkarte ihr eigen nennen, von Interesse sein könnte. Die Auflösungen dieser Monitore liegen denn auch teilweise deutlich über der des Amiga, was allerdings in der praktischen Anwendung keinerlei Probleme mit sich bringt.

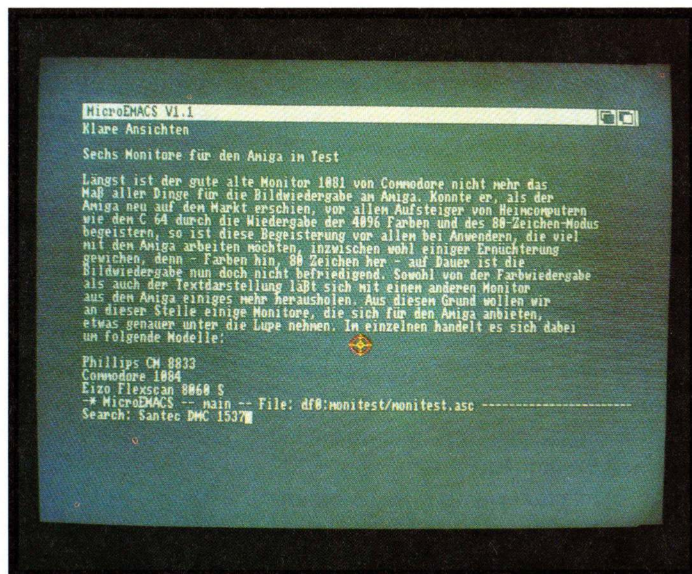
Kontaktschwierigkeiten

Bevor man in den Genuß der Anwendung eines Monitors kommt, gilt es zuerst einmal, diesen ordentlich mit dem Amiga zu verbinden. Keinerlei Probleme bereitet dies mit dem Monitor von Commodore, denn er wird - zumindest bei der in der Tabelle aufgeführten Bezugsquelle - mit dem passenden Kabel ausgeliefert, wobei erwähnt werden sollte, daß zusätzlich noch zwei Kabel - eines für die Verbindung mit PC-Rechnern und eines zum Anschluß an den Commodore 64 oder 128 - im Lieferumfang inbegriffen sind. Auch beim Mitsubishi muß man lediglich das Kabel einstecken, und schon kann's losgehen, denn der HF 1400 wird von der in der Tabelle genannten Vertriebsfirma mit Anschlußkabel speziell für Amiga ausgeliefert. Zusätzlich dazu ist er bereits leicht modifiziert, denn der HF 1400 läßt sich in der Form, wie er vom Hersteller ausgeliefert wird, nicht problemlos am Amiga betreiben.

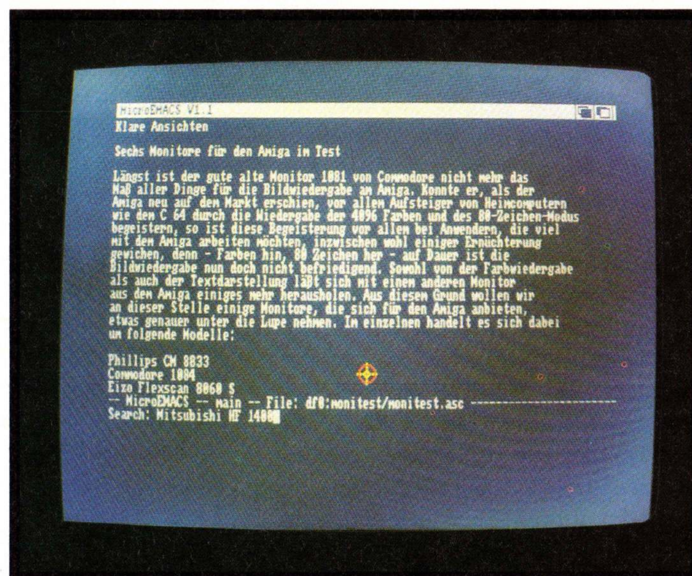
Bei den anderen Monitoren dieses Tests sind vor Inbetriebnahme erst einmal ein paar kleinere Hürden zu überwinden. Der Eizo und der NEC verfügen über einen 9-Pin-Eingang, während der Santec am Amiga mittels eines 15poligen Steckers betrieben werden muß. Der Phillips wiederum besitzt einen Euro-Scart-Eingang. Hier heißt es, sich im Fachhandel auf die Suche nach einem passenden Kabel zu machen, oder zu Einzelteilen und Lötkolben zu greifen. Für Anwender, die sich für letztere Möglichkeit entscheiden, sind die Pinbelegungen und Anschlußvorschriften abgedruckt. Bei jedem drei Multiscan-Monitore ist im Lieferumfang allerdings ein Kabel mit D-Sub 9Pol-Steckern enthalten, an das man, wenn man bereit ist, es zu opfern, nur einen bzw. beim Santec zwei neue Stecker anlöten muß. Diese sind im Elektronikfachhandel für wenig Geld erhältlich. Die technischen Einzelheiten der verschiedenen Monitore sind nicht in die nachfolgenden Beschreibungen der einzelnen Modelle mit eingeflossen; sie werden aufgrund der besseren Übersicht in der Tabelle "Technische Daten" aufgeführt.

Phillips CM 8833

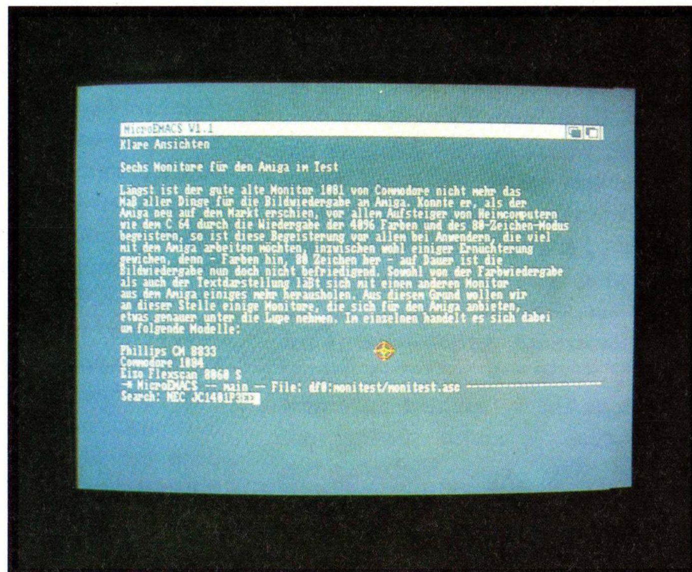
Der Phillips zählt zu den weniger teuren Monitoren des Testfelds. Dennoch hat er in einem - für den Amiga nicht unbedeutenden Punkt - die Nase vorn: Er verfügt nämlich über Stereowiedergabe des Tons. Allerdings



Textwiedergabe auf dem Santec DMC 1537



Textwiedergabe auf dem Mitsubishi HF 1400



Textwiedergabe auf dem NEC JC 1401 P3 EE

erweist sich die Schlitzmaske mit einer Rasterung von 0.42 mm gegenüber den teureren Modellen doch als recht grob. In der Bildwiedergabe ist der CM 8833 dem 1081 leicht überlegen; die Farben sind voller als beim 1081, und auch das Schriftbild ist feiner aufgelöst, sauberer und verfügt über mehr Kontrast. Dieser Monitor eignet sich universell für Grafik- wie Textwiedergabe und ist dem 1081 in jeder Hinsicht vorzuziehen, wenn auch im Vergleich mit den wesentlich teureren Multiscan-Monitoren Abstriche zu machen sind. Nach längerer Arbeit vor diesem Monitor - vor allem bei Textverarbeitung oder Programmentwicklung - stellen sich schon einmal Ermüdungserscheinungen ein, was allerdings in Anbetracht der Preisklasse des CM 8833 mehr oder weniger zu erwarten ist. Mit Farbkontrasten vor allem von Rot auf Blau und bei sehr feinen Linien hat er leichte Probleme, pixelbreite Linien neigen etwas zum Verschwimmen. Leider ist dieses Modell auch nicht entspiegelt, weshalb es größere Sorgfalt bei der Aufstellung verlangt, damit keine störenden Reflektionen den Spaß verleiden. Immerhin verfügt der Phillips über einen recht dunklen Schirm.

Der Interlace-Modus läßt sich mit diesem Monitor gut darstellen; die Bildwiedergabe übertrifft hier in der Qualität - in Hinsicht auf das berüchtigte Flackern - die des 1081. Von der Monitorseite her gesehen gibt es hier also keinen Anlaß zur Kritik.

Von der Justage her gab es beim Phillips wenig Anlaß zur Kritik; lediglich eine winzige Verbreiterung des Bildes im unteren Bereich war festzustellen.

Der CM 8833 verfügt sowohl über einen RGB- als auch über einen Videoeingang, womit er dem Benutzer die Möglichkeit bietet, ihn - mit einem TV-Tuner oder einem Videorekorder - auch als Fernsehschirm zu benutzen. Dazu muß man nicht einmal Kabel umstecken, denn die Eingänge sind an der Vorderseite umschaltbar. Über RGB harmoniert er außerdem sowohl mit einem Analog- als auch mit einem TTL-Signal. Weiterhin verfügt der Phillips über eine Vielzahl von Regelmöglichkeiten: horizontale Zentrierung, Helligkeit, Kontrast, Farbsättigung (nur über Videoeingang), Lautstärke und Monochromwiedergabe (in Grün) sind an der Vorderseite einstellbar. Auf der Rückseite des Phillips finden sich Regler für die vertikale Größe und Zentrierung sowie für die horizontale Größe und die Art des Eingangssignals bzw. die Art des RGB-Signals. Alles in allem bietet er sämtliche Regelmöglichkeiten, die man sich wünschen kann.

Commodore 1084

Für den Commodore 1084 gilt eigentlich all das, was schon über den Phillips gesagt wurde, denn diese beiden Monitore unterscheiden sich nur äusserst geringfügig. Dies verwundert allerdings wenig, da im Commodore-Gehäuse eine Bildröhre und Elektronik von Phillips stecken. Einer der Unterschiede dieser beiden Monitore besteht darin, daß der Commodore eine entspiegelte Bildröhre besitzt. Sein Farbwiedergabe erscheint allerdings einen Hauch blasser als die des CM 8833, was möglicherweise auf eben diese Entspiegelung und die damit verbundene Mattierung des Bildschirms zurückzuführen ist. Leider verfügt der 1084 nur über Mono-Tonwiedergabe. Ein weiterer kleiner Gegensatz besteht in den Bedienungselementen: Während der Commodore einen Monochrom-Modus vermissen läßt, verfügt er andererseits über einen zusätzlichen Bildschärferegler, der allerdings lediglich das Videosignal beeinflußt. Damit ist die Liste der Unterschiede aber auch schon voll, weshalb es sich erübrigt, dieses Modell noch weiter zu beschreiben.

Eizo Flexscan 8060S

Der Eizo Flexscan, den wir in Heft 12/87 bereits vorgestellt hatten, zählt zu den teureren Multiscan-Modellen. Er ist ein universell an den verschiedensten Rechnern einsetzbarer Monitor und erreicht eine Auflösung von maximal 820 mal 620 Bildpunkten. Mit einem Punktabstand von 0.28 mm verfügt er über die feinste Lochmaske aller Testgeräte. Dies schlägt sich durchaus in der Bildwiedergabe nieder, denn der Eizo liefert ein gestochen scharfes, sauberes Bild mit sehr hohen Farbkontrasten. Dies ist im Testbild vor allem bei den feinen Farbstreifen erkennbar. Unterschiede im Kontrast zwischen Bildrand und Bildmitte waren nicht feststellbar. Das Bild des 8060 S steht sehr ruhig und erzeugt ausgesprochen satte Farben; in letzterem Punkt übertrifft er die anderen Modelle, wobei es sich allerdings schon fast wieder um eine Geschmacksfrage handelt, ob man nun die eher satten oder die eher sanften Farbtöne vorzieht. Auf jeden Fall erweist sich der Eizo als optimal, wenn es darum geht, längere Zeit am Bildschirm zu arbeiten, denn die saubere Wiedergabe in Verbindung mit dem sehr ruhigen Bild, das fast keinerlei merkliches Zittern aufweist, schont gerade bei Textverarbeitung oder ähnlichem die Augen sehr. Er läßt sich übrigens neben dem Farbmodus in zwei Monochrom-Modi betreiben, wobei Schwarz-Weiß- sowie Amber-Wiedergabe zur Verfügung steht. Letztere erweist sich dem Grün-Modus des Commodore 1081 und des Phillips-Monitors überlegen, denn diese Farbeinstellung besitzt - bei sauberer Textdarstellung - geringere Kontraste, was die Augen angenehm entspannt. Allerdings ist dieser Unterschied wohl nur bei stundenlangem Arbeiten relevant, und erweist sich somit unter Umständen als Geschmacksfrage.

Leider verfügt der Eizo - wie übrigens alle Multiscan-Modelle im Test - über keinerlei Tonwiedergabe, was etwas unverständlich erscheint, auch wenn er nicht speziell für den Amiga gebaut ist, denn schließlich verfügen auch andere Rechner über Tonwiedergabe. Hier bleibt dem Besitzer nur die Möglichkeit, den Amiga an eine eventuell vorhandene Stereoanlage anzuschließen, oder sich kleine Aktivboxen, wie sie für diverse Walkman-Modelle angeboten werden, zuzulegen.

Das größte Manko des 8060 S am Amiga ist allerdings die Bildwiedergabe im Interlace-Modus, denn sie erweist sich als die schlechteste aller Testgeräte. Da sich der Interlace-Modus mit seinen zwei versetzen Halbbildern für die Synchronisationselektronik des Eizo als Unregelmäßigkeit darstellt, gibt diese ihr Bestes, um diese Unregelmäßigkeit auszugleichen. Dadurch geht der gewünschte Effekt des Interlace verloren; das Bild wird nicht in der vorgesehenen Form

überlagert und erweist sich als nahezu indiskutabel für fast jede Anwendung. Bei Grafik ist dieser Effekt unter Umständen zwar noch zu ertragen, aber Menüfunktionen in einem Grafikprogramm in Interlace sind extrem schlecht lesbar.
Die Bildröhre des Eizo ist dunkel und gut entspiegelt. Leider war das Testgerät nicht optimal justiert (die vertikalen Linien waren leicht ausgebeult), was man bei einem Gerät dieser Preisklasse eigentlich nicht erwarten sollte. Auch übersteuert der Kontrastregler, wenn er voll aufgedreht wird, und das Bild ist, auch bei hellster Einstellung, relativ dunkel, was allerdings eher kleine Schönheitsfehler sind, die bei der Anwendung nicht weiter stören, denn es lassen sich trotzdem sehr gute Einstellungen finden.
Die Regelmöglichkeiten des Eizo sind recht umfassend; so kann man an der Vorderseite die Bildhöhe und -breite (letztere allerdings nur in 4 Stufen), den Kontrast, die Helligkeit und den gewünschten Wiedergabemodus (Amber, Farbe, Schwarz-Weiß) einstellen. Auf der Rückseite finden sich zwei Regler für vertikale und horizontale Zentrierung sowie zwei Schalter für die Wahl des Eingangssignals (RGB-Analog oder TTL/TTL-Farbe). Ein solider dreh- und kippbarer Monitorständer ist im Lieferumfang des Eizo übrigens inbegriffen.

Santec DMC 1537

Auch bei diesem Modell handelt es sich um einen Multiscan-Monitor. Eine seiner Besonderheiten ist der flache Bildschirm, der fast keinerlei Wölbung aufweist. Wie auch der Eizo und der NEC eignet sich der Santec zum Anschluß an verschiedene Rechner. Die Rasterungsgröße seiner Lochmaske beträgt 0.31 mm, womit er in der Lage ist, bis zu 800 mal 600 Bildpunkte darzustellen. Die Bildwiedergabe des DMC 1537 ist dementsprechend sehr sauber und kontrastreich. Das Bild steht sehr ruhig; wie beim Eizo fällt das gelegentliche minimale Zittern kaum auf. Dies unterstützt entspanntes Arbeiten auch über längere Zeit. Die Farben des Santec wirken eher sanfter, was man in keinem Fall mit matt verwechseln darf, denn sie werden sauber dargestellt, wirken aber nicht ganz so satt. Dennoch hat man deutlich vollere Farben als beim Commodore 1081. Farbkontraste bereiten dem Santec ebenfalls keinerlei Schwierigkeiten, wie man auf dem Testbild erkennen kann. Auch konnten keine Unterschiede im Kontrast zwischen Bildrand und Bildmitte festgestellt werden. Ein weiteres besonderes Feature des DMC 1537 ist die Möglichkeit, die Rot-, Grün- und Blauanteile des RGB-Signals getrennt zu aktivieren bzw. zu desaktivieren. Dadurch hat man die Möglichkeit, zwischen den verschiedensten Monochrom-Modi auszuwählen, denn aus den verschiedenen Kombinationen ergeben sich sieben verschiedene Farben für den Monochrom-Modus. Auf diese Art und Weise muß sich der Anwender nicht festlegen, sondern kann ganz nach persönlichem Geschmack und Empfinden die angenehmste Einstellung auswählen. In dieser Hinsicht bietet der Santec einen Vorteil gegenüber den meisten anderen Geräten im Test.
Einen Toneingang sucht man am Santec leider vergeblich; auch hier muß sich der Anwender etwas einfallen lassen, um die Klänge, die sich aus dem Amiga herauslocken lassen, hörbar zu machen. Dies sollte allerdings - was für alle getesten Modelle ohne Tonwiedergabe gilt - kein Entscheidungskriterium sein, denn immerhin kommt es bei einem Monitor vor allem auf die Bildwiedergabe an.
Den Interlace-Modus bewältigt der DMC 1537 ohne weitere Probleme. Die Intensität des Flackerns ist dabei deutlich geringer als beim 1081; so kann man Grafikprogramme zeitweise durchaus in Interlace betreiben. Bei der Textwiedergabe wird es da schon eher kritisch; das berüchtigte Flackern läßt sich eben kaum unterdrücken. Dies stellt aber kein Manko dar, denn vom Einsatz des Interlace-Modus bei der Textwiedergabe sollte man in jedem Fall absehen. Positiv wirkt sich im Interlace die Möglichkeit aus, verschiedene Farben im Monochrom-Modus auswählen zu können, denn bei einigen dieser Farbeinstellungen läßt die Intensität des Flackerns deutlich nach.
Der Santec hat die dunkelste Bildröhre aller Testmodelle und ist sehr gut entspiegelt. Die Regler dieses Geräts sind optimal justiert; so neigt der Kontrast selbst bei voller Einstellung nicht zum Übersteuern, und die Helligkeitseinstellung arbeitet über einen sehr weiten Bereich, so daß selbst dunkle Bilder z.B. eines Malprogramms deutlich wiedergegeben werden können. Leider war aber auch der DMC 1537 nicht optimal justiert; das Bild war in der Mitte in horizontaler Richtung ganz leicht gestaucht. Dies ist - wie auch beim Eizo - zwar kein großes Manko, da ein Fernsehtechniker den Monitor problemlos nachjustieren kann, aber, wie schon gesagt, in dieser Preisklasse kann man eigentlich optimale Justage ab Werk verlangen.
An Regelmöglichkeiten läßt der Santec keinerlei Wünsche offen. An der Vorderseite befinden sich zwei Regler für Kontrast und Helligkeit, sowie drei kleine Schalter in den Farben Rot, Grün und Blau, mit denen sich die einzelnen Farbanteile des RGB-Signals schalten lassen. Auf der Rückseite verfügt der DMC 1537 über vier Regler für vertikale und horizontale Zentrierung und die Größeneinstellung in beiden Bilddimensionen, sowie über einen Wahlschalter für die Art des Eingangssignals (RGB-Analog oder - Digital). Man findet hier übrigens auch zwei Eingänge, nämlich die bereits weiter oben erwähnte 15polige Buchse für das Analog-Signal und einen D-Sub-9Pol-Eingang für das digitale Signal. Soweit dies aus der englischen Bedienungsanleitung hervorgeht, sollte es möglich sein, beide Eingänge zu belegen und über den Signalwahlschalter zwischen ihnen zu wechseln, was es ermöglicht, den Monitor an zwei Rechner (z.B. an einen Amiga 2000 und eine eingebaute PC-Karte mit Grafikkarte) gleichzeitig anzuschließen und nach Bedarf umzuschalten. Wie der Eizo wird auch der DMC 1537 mit einem stabilen dreh- und kippbaren Monitorständer ausgeliefert.

Mitsubishi HF 1400

Der Mitsubishi war mit Abstand das teuerste aller Testmodelle. Er zählt ebenfalls zu den Multiscan-Monitoren, wenngleich er nicht über eine ganz so große Frequenzvariabilität verfügt wie zum Beispiel der Santec oder der Eizo. Allerdings ist dieses Modell auch weniger zum Anschluß an andere Rechner

vorgesehen, denn es wird - wie bereits erwähnt - vom Vertrieb speziell für den Amiga modifiert und mit passenden Anschlußkabeln versehen. Ohne diese Modifikation wäre es nicht möglich, diesen Monitor am Amiga zu betreiben. Die Lochmaske des HF 1400 entspricht mit einer Rasterungsgröße von 0.31 mm der des Santec; über die maximale Spalten- bzw. Zeilenanzahl war leider keine Angabe zu finden, sie dürfte sich aber im Bereich von 800 mal 600 Bildpunkten bewegen. Die Farben des Mitsubishi sind sehr satt, und das Schriftbild ist scharf und sauber. Entspanntes Arbeiten über längere Zeiträume ist somit auch mit diesem Monitor gewährleistet. Bisweilen - vor allem bei weißem Hintergrund - fällt jedoch ein leichter Blaustich des Bilds auf. Etwas nachteilig ist auch der zum Rand des Bilds leicht nachlassende Kontrast; dies führt dazu, daß das Bild von der Kontrasteinstellung her nur sehr schwer zu justieren ist, was sich auf Dauer schon bemerkbar macht. Die Bildwiedergabe ist besonders ruhig; in diesem Punkt hat der HF 1400 gegenüber den anderen Modellen des Testfelds die Nase vorn. Dies liegt an einer technischen Besonderheit: Bei diesem Gerät handelt es sich nämlich um einen Monitor mit langer Nachleuchtdauer. Dies führt aber auch dazu, daß veränderte Bildbereiche sehr langsam auf dem Bildschirm verblassen, was bei Textverarbeitung nicht weiter stört, bei Spielen oder anderer bewegter Grafik allerdings das Problem aufwirft, daß bewegte Objekte Fahnen hinter sich herziehen. Dies wirkt bisweilen leicht irritierend, was auch für Requester oder sich schließende Fenster gilt, die nicht sofort verschwunden sind, sondern sich sanft ausblenden. Wie auch beim Santec lassen sich beim Mitsubishi die Rot-, Grün- und Blauanteile des RGB-Signals getrennt schalten, was die gleiche Vielfalt an für den Monochrombetrieb verfügbaren Farben ermöglicht.



Das Testbild auf dem Phillips CM 8833



Das Testbild auf dem Commodore 1084



Das Testbild auf dem Eizo Flexscan 80605

Auch bei diesem Gerät stellt sich dem Anwender das Problem der fehlenden Tonwiedergabe; improvisieren ist somit auch hier angesagt.

Ihren großen Vorteil kann die lange Nachtleuchtdauer des HF 1400 im Interlace-Modus voll ausspielen, denn hierbei sorgt sie dafür, daß fast jegliches Flackern vom Bildschirm verschwindet. Ohne Zweifel ist diese Wiedergabe herausragend gut; man ist förmlich versucht, Textverarbeitungen in Interlace auszuprobieren, denn sogar bei der Textdarstellung ist das Bild relativ sauber und ruhig, wenngleich immer noch ein gewisses Restflackern bleibt, was auf Dauer die Augen doch

belastet (außerdem sind die Schriftzeichen im Interlace-Modus extrem klein, so daß man sehr genau hinsehen muß, um einen Text zu lesen). Mit Mal- bzw. Konstruktionsprogrammen wie Deluxe Paint, Digipaint oder Aegis Draw Plus läßt sich mit diesem Monitor optimal im Interlace-Modus arbeiten; es ist ein Genuß, die volle Auflösung von 640 mal 512 Pixeln zu fahren, ohne daß ein feststellbares Flimmern auftritt.

Die Bildröhre des HF 1400 ist grau und entspiegelt. Auch war das Testgerät recht gut justiert, wenn es auch in der Horizontalen minimal gekippt erschien. In Hinsicht auf die Regelmöglichkeiten enttäuscht dieser Monitor allerdings: Die einzige Möglichkeit, die Bildwiedergabe von außen zu beeinflussen, besteht in einem Kontrastregler und den drei Schaltern für die Farbanteile des RGB-Signals an der Rückseite. Leider neigt der Kontrastregler sehr leicht zum Übersteuern. Dieser Mangel an Einstellmöglichkeiten fällt besonders in Hinsicht auf einen Helligkeitsregler negativ auf; man hat bei diesem Gerät keine großen Variationsmöglichkeiten in Bezug auf den persönlichen Geschmack. Auch die Möglichkeit, die Bildgröße zu verändern, sollte - wenn dies auch nicht elementar ist - eigentlich gegeben sein. Vorhanden sind die dafür nötigen Regler schon, aber leider muß man, um sie zu erreichen, das Gehäuse des Geräts öffnen und zum Schraubenzieher greifen. Dies ist zwar keine Arbeit, die einen ausgebildeten Techniker erfordert, aber wohl dennoch nicht jedermanns Sache.

NEC MULTISYNC

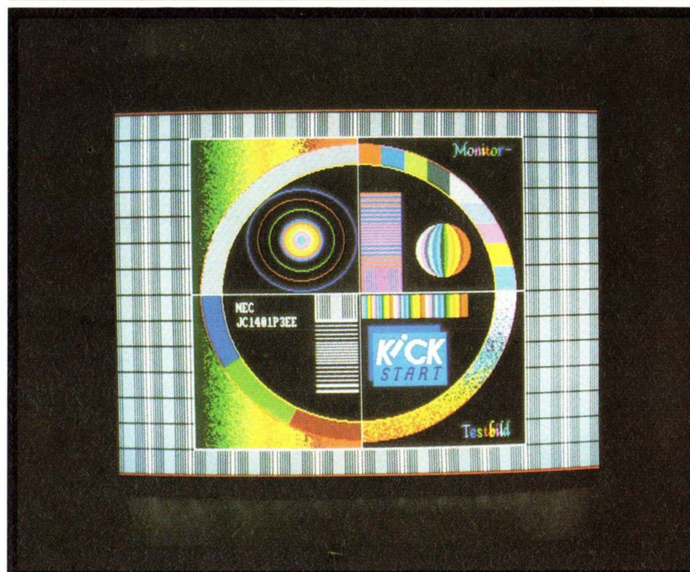
Der NEC Multisync ist wohl das bekannteste Modell aus der Gattung der Multiscan-Monitore. Obwohl schon seit einiger Zeit auf dem Markt, handelt es sich bei ihm um ein durchaus aktuelles Gerät, was auch gerade in Anbetracht des günstigen Preises, zu dem er teilweise erhältlich ist, gilt. Auch dieses Modell läßt sich an Rechner verschiedenster Bauart anschließen. Wie auch der Mitsubishi und der Santec verfügt der NEC über eine Lochmaske mit einer Rasterung von 0.31 mm; seine maximale Auflösung beträgt 800 mal 560 Bildpunkte. Das Schriftbild des Multisync ist sehr sauber und scharf; der Kontrast ist gut und weist keine Differenzen von Bildmitte zu Bildrand auf. Kontraste verschiedener Farben werden sauber dargestellt; Rot-Blau-Übergänge und feine Linien bereiten dem NEC keine Probleme. Auch dieser Monitor ist bei längerem Arbeiten am Bildschirm sehr



Das Testbild auf dem Santec DMC 1537



Das Testbild auf dem Mitsubishi HF 1400



Das Testbild auf dem NEC JC1401 P3 EE (Multisync)

entspannend, da auch das Bild sehr ruhig steht. In dieser Hinsicht entspricht er dem Santec und dem Eizo, denn wie bei diesen beiden Geräten ist nicht mehr als ein gelegentliches, fast unmerkbares Zittern bemerkbar. Eine weitere Gemeinsamkeit haben diese beiden Monitore in der Intensität der Farbwiedergabe, denn auch der NEC neigt zu eher sanfteren Farben. Laut Auskunft des Handbuchs verfügt der Multisync ebenso wie der Santec über die Möglichkeit, sieben verschiedene Farbkombinationen für die Textwiedergabe im Monochrom-Modus einzustellen; leider arbeitet diese Schaltung nur im TTL- und nicht im Analog-Modus, womit sie für reine Amiga-Anwender hinfällig wird. Wer aber noch über einen anderen Rechner oder eine 2000er-Karte mit RGB-Digital-Signal verfügt, kann in den Genuß der gebotenen Möglichkeiten kommen.
Daß die besseren Monitore nicht über Tonwiedergabe verfügen, scheint eine Regel zu sein, an die sich auch der NEC hält. Nun ja, man ist es inzwischen gewohnt und muß sich eben etwas einfallen lassen, wenn man zu gutem Bild und gutem Ton kommen möchte.
Im Interlace-Modus macht der NEC ein recht gutes Bild. Zwar kann auch er das berüchtigte Flackern nicht unterdrücken, aber für die Anwendung von Malprogrammen oder die Grafikwiedergabe in Interlace reicht es allemal. Hierbei erscheint die Wiedergabe um einiges ruhiger als beim 1081.
Der Multisync besitzt eine dunkle Bildröhre, und der Bildschirm ist sehr gut entspiegelt. Die Justage des Testgeräts war nahezu optimal; lediglich eine winzige Verschiebung im unteren Bildbereich in Richtung des rechten Bildrands war feststellbar, aber dies fiel nur bei sehr genauem Hinsehen auf.
Mit Regelmöglichkeiten ist der NEC gut ausgestattet; allerdings ist die Anordnung der Bedienungselemente doch etwas unorthodox. So befindet sich - im Gegensatz zu allen anderen Testgeräten - der Ein- und Ausschalter auf der Rückseite des Geräts. Die Regler für Kontrast, Helligkeit, vertikale Zentrierung und Größe, horizontale Zentrierung und vertikale Bildfrequenz (diese muß beim NEC von Hand justiert werden, was aber weder größere Geschicklichkeit erfordert noch irgendwelche Probleme aufwirft) sowie je ein Schalter für Textmodus (welcher sich, wie oben erwähnt, mit dem Amiga nicht anwählen läßt) und horizontale Größe findet man unter einer Klappe auf der Oberseite des Monitors. Leider ist die horizontale Bildgröße nur in zwei Stufen einstellbar. Auf der Rückseite finden sich noch ein Schalter für die Auswahl zwischen Analog- und TTL-Signal, ein Schalter für IBM-Modus (arbeitet nur mit dem TTL-Signal) sowie diverse Dipschalter, die ebenso alle der Beeinflussung des

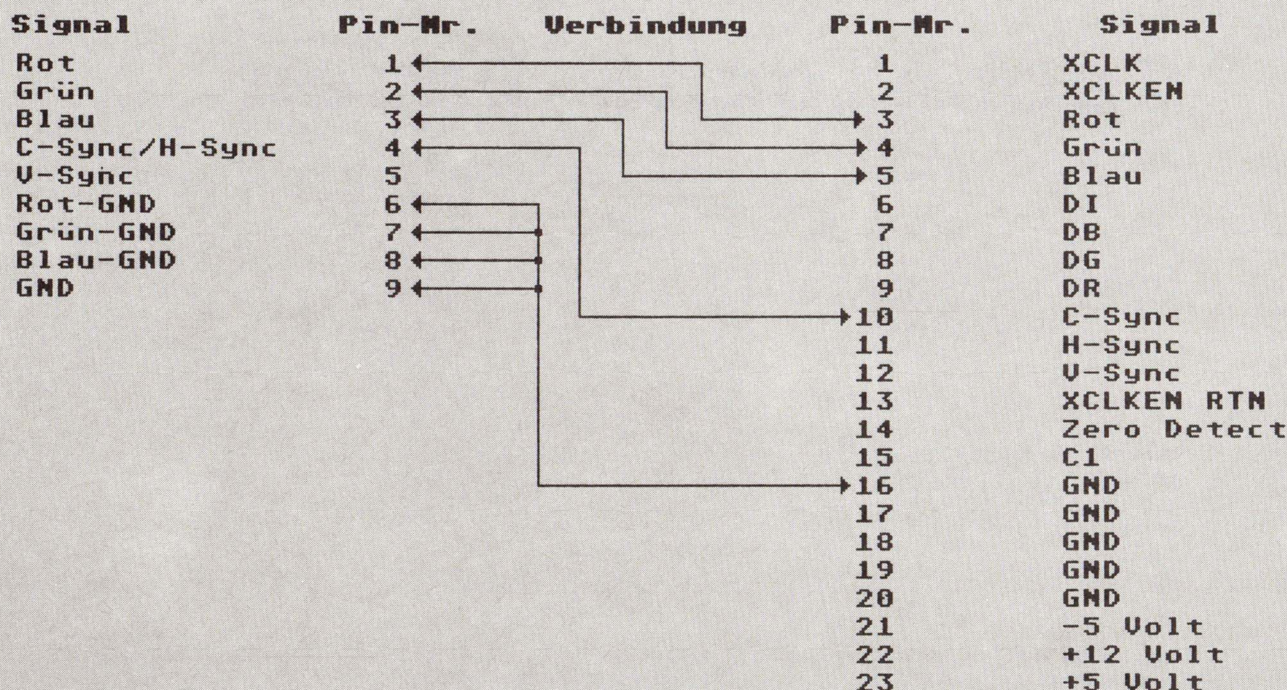
Verbindung des Santec DMC 1537 mit dem Amiga

Monitorseite — **D-Sub 15Pol-Stecker**

Rechnerseite — **D-Sub 23Pol-Stecker**

Signal	Pin-Nr.	Verbindung	Pin-Nr.	Signal
Rot	1		1	XCLK
Rot-GND	2		2	XCLKEN
Grün	3		3	Rot
Grün-GND	4		4	Grün
Blau	5		5	Blau
Blau-GND	6		6	DI
GND	7		7	DB
PGA Mode Control	8		8	DG
C-Sync	9		9	DR
Unbenutzt	10		10	C-Sync
Unbenutzt	11		11	H-Sync
Unbenutzt	12		12	V-Sync
Unbenutzt	13		13	XCLKEN RTN
H-Sync	14		14	Zero Detect
V-Sync	15		15	C1
			16	GND
			17	GND
			18	GND
			19	GND
			20	GND
			21	-5 Volt
			22	+12 Volt
			23	+5 Volt

Die GND-Pins 2,4,6 und 7 der Monitorseite können auch in beliebiger Kombination einzeln mit den GND-Pins 16-20 der Rechnerseite verbunden werden.



TTL-Signals vorbehalten sind. Ein Monitorständer erweist sich für den NEC als überflüssig, da das Gerät über einen ins Gehäuse integrierten Dreh- und Schwenkfuß verfügt.

Die Qual der Wahl

Will man sich einen anderen Monitor als den 1081 zulegen, so steht man vor einer schwierigen Entscheidung. Hierbei sind wohl viele Faktoren wie persönlicher Geschmack, bevorzugte Anwendung und nicht zuletzt auch der Inhalt des Geldbeutels in Betracht zu ziehen. Generell bieten die Monitore Phillips CM 8833 und Commodore 1084 für wenig Geld eine deutlich bessere Bildwiedergabe als der 1081 und sind für den nicht allzu anspruchsvollen Anwender als günstige Allround-Monitore zu empfehlen. Wer allerdings viel Zeit vor seinem Amiga verbringt und sich dann auch noch viel mit Textverarbeitung oder Programmierung beschäftigt, dem sei zu einer größeren Investition in Form eines der Multiscan-Modelle geraten, denn erst diese gewährleisten eine wirklich saubere Bildwiedergabe und somit augenschonendes Arbeiten. Aber auch hier ist es nicht leicht, sich festzulegen, denn jedes der Modelle hat seine mehr oder weniger bedeutenden Vor- und Nachteile. So erscheint der Eizo den Mitstreitern in Hinsicht auf Schriftbild und Bildruhe ganz leicht überlegen, schneidet aber im Interlace-Modus ausgesprochen schlecht ab. Andererseits bietet er auch die satteste Farbwiedergabe aller Testgeräte. Der Santec und der NEC sind sehr gute Monitore für die verschiedensten Anwendungen und bringen problemloses Arbeiten in jedem Modus mit sich. Sie sind für jeden Anwender, der einerseits viel am Gerät arbeiten und andererseits nicht auf gelegentliche Anwendung des Interlace verzichten möchte, beide absolut empfehlenswert. Allerdings sind ihre Farben nicht ganz so satt wie die des Eizo oder auch des Mitsubishi. Letzterer wiederum bietet dem Anwender die Möglichkeit, zumindest bei Grafikwiedergabe den Interlace-Modus problemlos und auch über längere Zeit anzuwenden, was sich für Anwender anbietet, die sich bevorzugt mit Mal- und Konstruktionsprogrammen beschäftigen. Allerdings hat dieses Gerät Probleme bei der Darstellung bewegter Bilder, weshalb es für Animationen und Spiele wenig geeignet ist. So bleibt dem Interessierten nur zu raten, anhand seiner bevorzugten Anwendungen eine nähere Auswahl zu treffen und sich diese Geräte dann - wenn möglich - im Fachhandel selbst einmal anzuschauen. In Anbetracht der Tatsache, daß man - wie einige Testgeräte zeigten - immer einmal mit kleinen Justierungsfehlern zu rechnen hat, erweist sich dies in jedem Fall als empfehlenswert. Wer diesen Weg nicht auf sich nehmen möchte und sich

eventuelle Sonderangebote zu nutze machen möchte, sollte sich anhand der hier gebotenen Entscheidungshilfen sehr gut überlegen, welches Gerät für ihn die beste Wahl darstellt.

Technische Daten

	Commodore 1084	Phillips CM 8833	Eizo Flexscan 8060 S
Bildröhre	14 Zoll dunkler Schirm entspiegelt Schlitzmaske	14 Zoll dunkler Schirm Schlitzmaske	14 Zoll dunkler Schirm entspiegelt Lochmaske
Ablenkung	90 Grad	90 Grad	90 Grad
Bildschirm größe	0.42 mm	0.42 mm	0.28 mm
Auflösung			
Zeilen	600	600	620
Spalten	k.A.	k.A.	820
Videosignal			
-Anstiegszeit	k.A.	k.A.	15 ns
-Abfallzeit	k.A.	k.A.	15 ns
Bildfrequenz	50/60 Hz	50/60 Hz	50 - 80 Hz (autom.)
Zeilenfrequenz	15.625 kHz	15.625 kHz	15.7 kHz - 35 kHz (autom.)
Eingangssignale	Composite Video RGB Analog/TTL	Composite Video RGB Analog/TTL	RGB Analog RGB Digital
Synchronisation			
Separat	k.A.	k.A.	positiv/negativ
Composite	k.A.	k.A.	positiv/negativ
Composite auf Grün	k.A.	k.A.	negativ
farbanzahl			
RGB TTL	k.A.	k.A.	8/16/64 Farben
RGB Analog	unbegrenzt	unbegrenzt	unbegrenzt
Bildeingänge	Euro-Scart DIN	Euro-Scart DIN	D-Sub 9Pol
Leistungs- aufnahme	75 Watt	75 Watt	88 Watt
Gewicht	11 kg	11 kg	13 kg
Preis	DM 750.-	DM 750.-	DM 2274.-
Bezugsquelle	PDC 6380 Bad Homburg 06172/24748	PDC 6380 Bad Homburg 06172/24748	Rein Elektronik 4054 Nettetal 02153/733-0

Technische Daten

	Santec DMC 1537	Mitsubishi HF 1400	NEC Multisync JC-1401P3EE
Bildröhre	15 Zoll, flach dunkler Schirm entspiegelt Lochmaske	14 Zoll, lange Nachleuchtdauer entspiegelt Lochmaske	13 Zoll dunkler Schirm entspiegelt Lochmaske
Ablenkung	90 Grad	k.A.	90 Grad
Punktraster-größe	0.31 mm	0.31 mm	0.31 mm
Auflösung			
Zeilen	600	k.A.	560
Spalten	800	k.A.	800
Videosignal			
-Anstiegszeit	k.A.	12 ns	k.A.
-Abfallzeit	k.A.	12 ns	k.A.
Bildfrequenz	50 - 90 Hz (autom.)	40 - 70 Hz (autom.)	56 - 62 Hz (manuell)
Zeilenfrequenz	15.5 kHz - 37 kHz (autom.)	15.5 kHz - 20.0 kHz (autom.)	15.5 kHz - 35 kHz (autom.)
Eingangssignale	RGB Analog RGB TTL	Composite Video Composite synchron. Separat synchron. Video	RGB Analog RGB TTL
Synchronisation			
Separat	positiv/negativ	k.A.	positiv/negativ
Composite	positiv/negativ	k.A.	positiv/negativ
Composite auf Grün	k.A.	k.A.	negativ
Farbanzahl			
RGB TTL	8/16/64 Farben	k.A.	8/16/64 Farben
RGB Analog	unbegrenzt	unbegrenzt	unbegrenzt
Bildeingänge	D-Sub 9Pol D-Sub 15Pol	Anschlußkabel an Amiga	D-Sub 9Pol
Leistungs-aufnahme	100 Watt	60 Watt	78 Watt
Gewicht	15.5 kg	12 kg	15.2 kg
Preis	DM 2277.-	DM 2998.-	DM 2164.-
Bezugsquelle	Sanyo Video Vertriebs GMBH 2070 Ahrensburg 04102/4901-0	Videocomp 6000 Frankfurt 069/467001	NEC Deutschland 8000 München 089/9099930

VON JÜRGEN SCHMIDT

KLAPPE, AUFNAHME, SCHNITT!

The Director nennt sich ein Programm, das zum Erstellen von Diashows und Video-Präsentationen dienen soll. Im Unterschied zu anderen Programmen dieser Art verzichtet der Director auf eine Benutzeroberfläche. Die Maus kann also getrost beiseite gelegt werden, hier ist der Programmierer gefragt, obwohl der Hersteller selbst der Auffassung ist, daß der Anwender noch nicht einmal etwas von BASIC wissen muß, um mit dem Programm umgehen zu können.

Nun, da sich die hier entworfene Sprache sehr stark an BASIC anlehnt, in manchen Befehlen sogar konform mit ihr ist, muß diese Behauptung doch zumindest bezweifelt werden. Es kann aber wohl davon ausgegangen werden, daß die meisten Amiga-Besitzer mit den Eigenheiten von Basic vertraut sind, so daß sich keine großen Schwierigkeiten im Umgang mit dem Programm ergeben dürften.

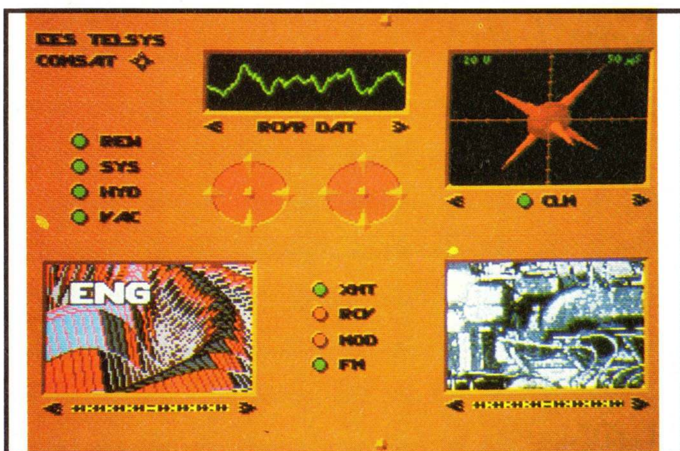
Unterschiede zu BASIC liegen vor allem in der Erweiterung des Befehlssatzes für den grafischen Bereich. Das Programm stellt verschiedene Anweisungen zum Laden und Zeigen von Files im IFF-, ANIM- und IFF-ILBM-Standard zur Verfügung. Somit können fast alle Bilder der besten und gebräuchlichsten Grafikprogramme wie Videoscape 3D, Dpaint, DigiPaint oder Aegis Images bearbeitet werden. Der Gebrauch verschiedener Fonts im Amiga-Format sowie der Soundfähigkeiten des Rechners wird genauso unterstützt wie die relativ einfache Verwaltung von Bildschirm und Speicher. Bilder und Animationen können in sogenannten Puffern, also Speichern im Speicher, abgelegt werden, von denen das Programm so viele besitzt, wie vom Anwender zugewiesen werden, was nur vom vorhandenen Speicherplatz begrenzt ist. Das Programm arbeitet nämlich vorzüglich mit dem Fast-RAM zusammen. Der Copy-Befehl ersetzt hier den Blitter, der ja bekanntlich mit dem Fast-RAM nichts anzufangen weiß, ansonsten aber gute Dienste tut, wenn es darum geht, große Datenmengen umherzuschaufeln. Diese Dienste kann der Benutzer sich gezielt zu eigen machen, mit Blit, Blitdest und Blitmode stehen drei außerordentliche Befehle zur Verfügung, mit denen nicht nur das schnelle Verschieben von Bildteilen über den Screen oder innerhalb des Speichers möglich ist. Weitere Anwendungen wären z.B. Mischen verschiedener Bilder im Speicher, invertieren eines Objekts zum Negativ, oder Pageflipping in mehreren Windows auf einem Screen zur selben Zeit, um nur einige zu nennen, denn die hier gebotenen Möglichkeiten sind wirklich vielfältig. Allein Blitmode kennt 256 verschiedene Zustände, von denen zwar nicht jeder immer sinnvoll ist, was aber dennoch eindrucksvoll die Vielfalt der Bildmanipulationen, die hier angeboten werden, dokumentiert.

Mit dem Planes-Befehl werden die unterschiedlichen Bitplanes aktiviert, so daß die oben beschriebenen Anweisungen, sowie die Zeichen-Befehle wie Draw, Ellipse, Circle, Rect oder Move sich nur auf dem gerade aktuellen Bitplane vollziehen. Selbst die Bildschirmauflösung eines Puffers kann vom Benutzer selbst bestimmt werden, somit sind sogenannte Diashows mit unterschiedlich aufgelösten Bildern keine Unmöglichkeit mehr. Schon selbstverständlich ist die Definition von verschiedenen Blenden wie Wipe, Fade, Flip und Dissolve.

Ansonsten sind alle notwendigen Deklarations- und Steueranweisungen vorhanden, die BASIC kennt, um dem Programm den gewünschten Verlauf zu geben. Das Programm beinhaltet so viele Features, daß eine komplette Aufzählung mit dem Abdruck des



Mit "THE DIRECTOR" zurück in die 50er.



4 Fenster auf einem SCREEN; natürlich animiert.

umfangreichen Handbuchs gleichzusetzen wäre.

THE DIRECTOR lief auf allen PAL- und NTSC -Amigas mit einem Laufwerk und mindestens 512 KByte, doch empfehlenswert wäre eine Speicherkonfiguration mit 1 MByte. Das Handbuch ist gut gegliedert und dürfte dank einiger Beispiele und Illustrationen auch für die Neulinge der Amiga-User ausreichen, zumal ein ausführliches BASIC-Handbuch im Lieferumfang des Amiga beinhaltet ist. Noch ein Tip zum Schluß, schauen Sie sich einmal Probe Sequence an, ein Demoprogramm, das mit THE DIRECTOR erstellt wurde und einen Eindruck der immensen Möglichkeiten dieser Software vermittelt.

The Director
Herst.: The Right Answers Group
Preis: ca. 150.- DM
Vertrieb: Fachhändler

VON JÜRGEN SCHMIDT

TITEL IM LAUFSCHRITT

Oder: Die Bilder laufen schon lange, doch wie sieht es mit den Titeln aus?

Bisher mußten sie mühsam mit dafür gar nicht vorgesehenen Malprogrammen erstellt werden. Doch damit soll jetzt Schluß sein. VIDEO-TITLER heißt das Programm, das aus dem Dilemma komplizierter Titelherstellung für Videos und Präsentationen helfen soll.

Hergestellt wurde das Softwarepaket von AEGIS, nach deren Philosophie dieses Programm die Lücke zwischen den einfachen Textgeneratoren der Hobby-Videofilmer und den sehr teuren Grafik- und Animationsgeräten der Profis schließen soll. Also gute Qualität zu einem Preis, den man sich leisten kann. Doch vor diesem Programm sollte man sich schon ein paar Dinge geleistet haben, als da wären ein zweites Laufwerk sowie eine Speichererweiterung auf 1,5 Megabyte. Mit einem Megabyte kann auch gearbeitet werden, wenn man auf Medres. und Highres. mit Overscan verzichtet. Besitzt man 2 Megabyte bzw. eine Festplatte, so stehen dem Anwender alle erdenklichen Möglichkeiten, die dieses Programm bietet, zur Verfügung. Soviel zur erforderlichen Konfiguration.

Das Softwarepaket besteht aus der Programm- und der Datendiskette sowie zwei Handbüchern, dies aus dem einfachem Grund, daß sich auf der Programmdiskette auch zwei Programme, VIDEO-TITLER und VIDEOSEG, befinden.

Um gute Titel zu erzeugen, bedarf es verschiedener Schriftarten und Stilrichtungen. Hier steht dem Anwender eine große Auswahl zur Verfügung, denn es kann mit jedem Font im Amiga-Format gearbeitet werden. Andere Fonts als die von der Datendiskette erreichbaren können einfach im Arbeitsspeicher installiert werden, wo maximal zehn Fonts zu einem Zeitpunkt ansprechbar sind. Jeder Font kann mit einer von zwanzig Stilrichtungen, wie z.B. Schatten, 3D-Backdrops, Neon oder Umrandungen, effektvoll kombiniert werden. Weiterhin können nicht nur Buchstaben- und Schattenfarbe beeinflußt werden, sondern auch die Tiefe und Richtung der Schatten und der 3D-Backdrops obliegt der vollständigen Kontrolle des Benutzers. Wem dies allerdings noch nicht genug ist, der kann sich im sogenannten Expert-Mode seinen eigenen Stil zulegen. Doch wer meint, hiermit sei das Thema Fonts nun abgeschlossen, der irrt, denn es gibt noch die sogenannten Poly-Fonts. Wird diese Option angewählt, so erscheint der eingegebene Text in einem Fenster, welches verkleinert oder vergrößert bzw. schräggestellt oder gekippt werden kann, so daß der Text z.B. auf den Kopf gestellt erscheint. Poly-Font hält noch einmal fünf weitere Schriftarten bereit, aber auch die Amiga-Fonts können als Poly-Fonts bearbeitet werden, so daß jeder Schriftzug in verschiedenen Größen darstellbar ist.

Ediert wird mittels eines Fadenkreuzes, dessen Mittelpunkt einfach an die entsprechende Stelle des Bildschirms gebracht wird, woraufhin der Text eingegeben, verändert, gelöscht oder auf eine neue Position gesetzt werden kann. Als Hintergrund für ein Titelbild eignet sich jedes IFF-Bild. Die Möglichkeiten der Screenmanipulation sind dabei mannigfaltig, so bestehen Optionen zum Bestimmen von Vordergrund und Hintergrund, Mischen der beiden Screens, sowie Schneiden und Einfügen mittels eines Zwischenspeichers. Über das Clipboard können nicht nur ganze Screens, sondern auch Windows oder Brushes von mehreren verschiedenen IFF-Bildern auf einen gemeinsamen Hintergrund gesetzt werden. Farbverlauf, Spiegelung enlang der X- bzw. Y-Achse, Komprimieren, Strecken und Transparentmodus sind weitere nützliche Features, die genutzt werden können.

Mit einigem Geschick und Übung lassen sich sogar 3D-Effekte oder z.B. das Füllen von Buchstaben mit einem bestimmten Untergrund erreichen. Sollten die Möglichkeiten der Bildgestaltung einmal nicht ausreichen, kann der Anwender dank des IFF-Standards das Bild zur Weiterbearbeitung auch in andere Malprogramme einladen.

Unterstützt werden alle verschiedenen Auflösungen mit Ausnahme des HAM-Modus. Auch Overscan und eventuell vorhandene Halfbrite-Fähigkeiten sind im Programm voll implementiert.

Halfbrite verdoppelt die Farbpalette, ist aber nicht auf allen Amiga 1000-Modellen nutzbar. Grund hierfür sind unterschiedliche Versionen des Customchips Paula. Keine Probleme dürften sich allerdings bei den Amiga 500- bzw. Amiga 2000-Rechnern ergeben. Zur Sicherheit ist ein Halfbrite-Test mit im Programm eingefügt.

Sind nun auf die eine oder andere Art und Weise Bilder und Text erstellt worden, so geht es jetzt an die Bearbeitung einer Animation. Die Prozedur dazu ist denkbar einfach. Zuerst werden die Bewegungskriterien festgelegt, die Bewegung in ihre Einzelschritte unterteilt und das fertige Bild dann in einem Anim-File abgespeichert. Dieser Vorgang wird fortgesetzt, bis die gewünschte Bewegung abgeschlossen ist. Danach wird der Anim-File geschlossen, und fertig ist die Animation, die jetzt jederzeit zum Abspielen bereit ist. Dies geschieht, wie Sie sich sicherlich schon gedacht haben, mittels Page-Flipping, also durch schnelles Zeigen der aufeinanderfolgenden Bilder. Da der Text sowie die Inhalte des Clipboards wirklich an jede Stelle des Bildschirms positioniert werden können, sind somit sehr fließende Bewegungen möglich, was natürlich auch eine erhöhte Anzahl von Bildern bzw. benötigtem Speicherplatz zur Folge hat. Um diesen Umstand soweit wie möglich einzuschränken, wurde auf eine Kompressionstechnik zurückgegriffen, welche nur die Unterschiede zwischen den einzelnen Bildern berechnet und speichert.

Werden allerdings sehr viele, oder große Objekte, gleichzeitig über den Schirm bewegt, so spiegelt sich das nicht nur in längeren Files, sondern auch in einer verlangsamten Animationsgeschwindigkeit wieder.

Doch um einen Vorspann für einen Videofilm oder gar eine Präsentation zu erstellen, ist noch etwas mehr von Nöten als einfaches Page-Flipping.

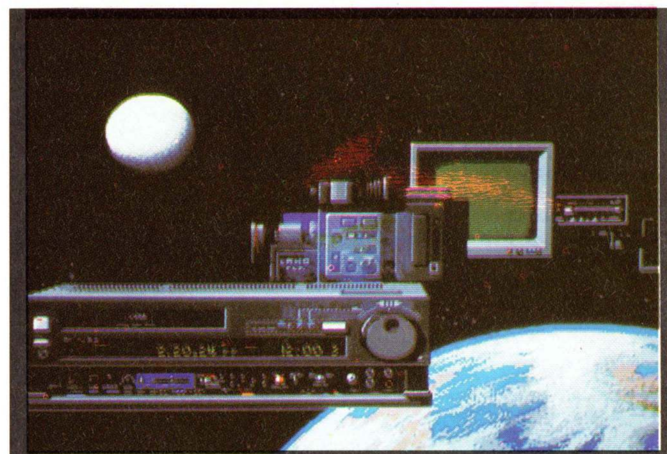
An dieser Stelle setzt VIDEO-SEG an. Der Name steht schlicht und ergreifend für Video Special Effect Generator. Aber keine Angst vor großen Namen, das Programm erwies sich in seiner Handhabung ebenso einfach wie nützlich. Konstruiert wurde VIDEO-SEG hauptsächlich für die Zusammenarbeit mit dem VIDEO-TITLER, da aber auch hier mit dem IFF und ANIM-Standard gearbeitet wird, steht einer Kooperation mit anderen Grafik- und Animationsprogrammen nichts im Weg. Die hiermit zusammengestellten Diashows werden als Script gespeichert und sind damit jederzeit wieder veränderbar. In diesem Script finden sich nicht nur alle nötigen Informationen über die unterschiedlichen Bilder und Animationen, sondern auch die verschiedenen Übergänge oder Blenden zwischen einzelnen Bildern sind hier definiert. So kann der An-



Font as font can.



Neonschriften sind kein Problem.


IFF- Bilder können als Hintergrund geladen werden.

wender zwischen Flip, Fade, Dissolve, Diagonals, Diamonds, 9 verschiedenen Wipes und 10 unterschiedlichen Blockoperationen die Art des Bildaufbaus selbst bestimmen. Fade blendet das aktuelle Bild aus, um kurze Zeit später das neue Bild einzublenden. Bei den Blockoperationen wird der Screen in 144 Felder unterteilt, somit sind z.B. spiralförmige Bildaufbauten von innen nach außen oder auch umgekehrt möglich. Dissolve bietet wohl eine der schönsten Formen des Bildwechsels, die mit diesem Programm möglich sind. Dabei wird von Pixel zu Pixel das eine Bild aus- und das nächste gleichzeitig eingeblendet, so daß man den den Eindruck gewinnt, der Screen würde sich auflösen. Da jedesmal die Farbpalette mit dem Bild gewechselt wird, bietet VIDEO-SEG auch eine Möglichkeit, diese Überblendugen auf eine einfache Art zu testen, bevor man einen Script schreibt. Alle diese Aktionen sind nur zwischen zwei Einzelbildern anwendbar. Die in einem Script eingebundenen Animationen sind verständlicherweise davon nicht betroffen.

Um bei längeren Filmen mit vielen Bildern und Animationen Verzögerungen durch lange Ladezeiten von der Floppy zu vermeiden, können Bilder sowie ANIM-Files in 99 verschiedene Buffer geladen und von dort aus gezeigt werden. So wird z.B. Screen 1 und 2 geladen, und während die Bilder für eine selbst zu bestimmende Zeit gezeigt werden, kann ein ANIM-File in den Speicher geladen und anschließend von dort aus abgespielt zu werden.

Während dieser Zeit können nun wiederum die Puffer, die Bild 1 und 2 enthalten, gelöscht und mit neuen Daten belegt werden, so daß der Benutzer, auch bei begrenztem Speicherplatz, mit sorgfältiger Planung lange Präsentationen ohne störende Verzögerungen erstellen und zeigen kann. Positiv hierbei ist auch, daß sämtliche mit VIDEO-TITLER und VIDEO-SEG erarbeiteten Animationen oder Scripts auch vom Cli oder einem Execute-File aus aufrufbar sind, was wiederum wertvollen Speicherplatz spart.

Beide Programmme laufen in NTSC- und PAL-Auflösung. Ein Manko sind die etwas knapp gehaltenen Handbücher, die zwar auf alle Befehle kurz eingehen, aber hier mangelt es an praktischen Beispielen und Tips, so daß der Benutzer, trotz einfacher Bedienung des Programms, zum Experimentieren gezwungen ist. Trotzdem ist der VIDEO-TITLER ein von seiner Anwendung her nützliches Programm, mit dem sich gerade wegen seiner Übersichtlichkeit und, wie gesagt, einfachen Handhabung angenehm arbeiten läßt. Leider können keine Bilder im H.A.M.-Modus verarbeitet werden.

Anzumerken wäre noch, daß die Handbücher wieder einmal nur in englischer Sprache erhältlich sind.

Videotitler
Herst.: Aegis Developement
Preis: ca. 280.- DM
Vertrieb: Fachhändler

WORT PERFEKT:

BECKERtext Amiga



Das deutsche Textwunder

Alle, die viel schreiben, brauchen eine Textverarbeitung, die alles kann und trotzdem schnell und komfortabel ist. Denn was nützt der größte Leistungsumfang, wenn man die Vielfalt der Funktionen nicht im Kopf hat und immer wieder das Handbuch wälzen muß?
Nein, eine Alleskönner-Textverarbeitung muß her. Mit allen Features, die man wirklich braucht, der vollen Integration in die AMIGA-INTUITION-Oberfläche – sprich: Anklicken aller Befehle mit der Maus – und dazu noch ein akzeptabler Preis. Wunschtraum oder Realität?
Die Antwort heißt BECKERtext AMIGA.

Schnelle Direktformatierung:

WYSIWYG-Prinzip: keine störenden Steuerzeichen im Text, schnelle Direktformatierung am Bildschirm mit allen Attributen (fett, kursiv, unterstrichen, Blocksatz, zentriert, linksbündig, rechtsbündig, hochstellen, tiefstellen, Horizontal- oder Vertikaldruck, Variation der Zeichendichte).

Einbindung von Grafiken:

Wenn schon AMIGA, dann auch eine Textverarbeitung, die Grafiken verarbeitet.
Für BECKERtext kein Problem: Das integrierte Hilfsprogramm BTSnap kann alle Grafiken im IFF-Format (Dateiformat, mit dem fast alle Mal- und Zeichenprogramme für den AMIGA arbeiten) und Bildschirmausschnitte der Workbench einlesen. Eine starke Sache.

Rechnen im Text:

Eine Textverarbeitung soll souverän mit Worten operieren, aber wie ist es mit Zahlen? Für BECKERtext AMIGA eine Leichtigkeit: Rechnen im Text, sowohl spalten- als auch zeilenweise. Mit bis zu 6 Nachkommastellen und 10stelliger Genauigkeit. Selbstverständlich mit Dezimaltabulator. Ein besonderer Vorteil für die Tabellenverarbeitung.

Formulare nach Wahl:

Mit BECKERtext AMIGA können Sie beliebige Formulare definieren und bis auf Abruf speichern (z. B. für Rechnungen, Lastschriftformulare, Tabellen, Briefpapier, Seitenlayout, etc.). Die lästige Neudefinition bewährter Standardformate entfällt – wieder ein Pluspunkt mehr.

Elektronische Rechtschreibhilfe:

Normalerweise folgt jeder Texteingabe die Korrektur. BECKERtext AMIGA leistet Vorarbeit: Das integrierte ONLINE-Lexikon überprüft den Text schon während der Eingabe auf Fehler in der Rechtschreibung (wahlweise auch danach). Da es individuell erweiterbar ist, eignet es sich auch für Fremdsprachen.

Überlegene Features:

Mehrspaltige Druckausgabe. Beim Ausdruck können Textdateien miteinander verknüpft werden. Multitasking: paralleles Arbeiten mit mehreren Programmen in verschiedenen Fenstern. Von einer Vorlage können bis zu 99 Kopien nacheinander ausgedruckt werden. Dreifache Funktionstastenbelegung mit maximal 160 Zeichen zur Speicherung von Floskeltexten oder Tastaturmakros. 1- und 2-bahniger Etikettenausdruck. Automatisches Erstellen von Stichwort- und Inhaltsverzeichnissen. Serienbrieffunktion mit Übernahmemöglichkeit aus beliebigen ASCII-Dateien. Datentransfer über RS 232. Umfangreiche Blockoperationen (Suchen, Ersetzen, Kopieren, Verschieben). Komfortable Druckeranpassung mit integriertem Treiber für alle gängigen Drucker. Querdruck auf Epson-kompatiblen Druckern bis zu 999 Zeichen pro Zeile. Ausführliches deutsches Handbuch. Minimalkonfiguration: 1 MByte RAM.

BECKERtext AMIGA
nur DM 199,–

DATA BECKER
Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

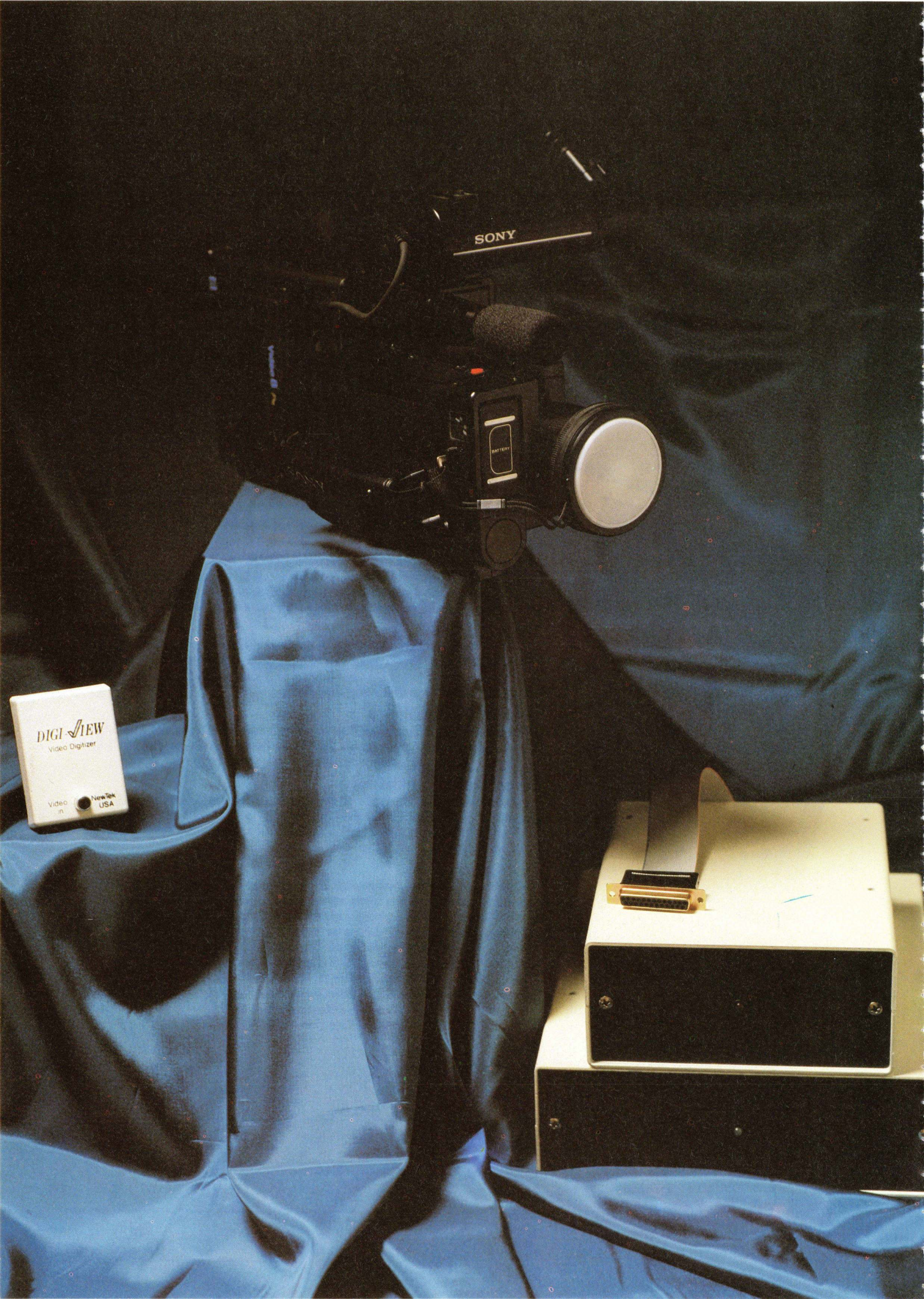
COUPON

COUPON BITTE EINSENDEN AN:
DATA BECKER
MEROWINGERSTR. 30
4000 DÜSSELDORF

HIERMIT BESTELLE ICH

NAME, VORNAME

STRASSE, ORT


SONY
BATTERY
DIGI-VIEW
Video Digitizer
Video in
NewTek USA

VON WOLF DIETRICH

AUF BILDFANG

Digitalisierte Bilder haben auf dem Amiga sehr schnell eine immer größere Verbreitung bekommen, vor allem in Diashows, aber auch als Hintergrundgrafiken in verschiedenen Arten von Programmen. Für den in dieser Richtung ambitionierten Grafikfreund bietet sich eine immer größere Palette von auf dem Markt befindlichen Digitizern an. Wir haben aus diesem Angebot zwei Produkte herausgegriffen, nämlich den bereits seit längerer Zeit erhältlichen und sehr preisgünstigen DigiView von NewTek und den wesentlich teureren Merkens VD 3 AMIGA. Zusätzlich zu diesen beiden Digitizern hatten wir noch einen RGB-Splitter VD 3 AMIGA von Merkens im Test, welcher in Verbindung mit beiden Digitizern genutzt wurde.

Die beiden Gerät unterscheiden sich schon äußerlich sehr stark. Während sich DigiView als winziges Gerät, kaum breiter als der Parallelport, auf den er aufgesteckt wird, darstellt, handelt es sich beim VD 3 AMIGA um ein Kästchen in der ungefähren Größe einer original Commodore-Zweit-floppy. Aber auch in den Eingängen unterscheiden sich die Geräte: Während der DigiView über nur einen Eingang für ein Videosignal verfügt, hat der Merkens derer gleich vier, von denen eines noch dazu mit einem Chroma-Filter versehen ist, der einen problemlosen Anschluß einer Farbkamera ermöglichen soll. Weiterhin gibt es am VD 3 einen Triggeranschluß, an den ein externer Auslöser angeschlossen werden kann.

Der Farbspalter

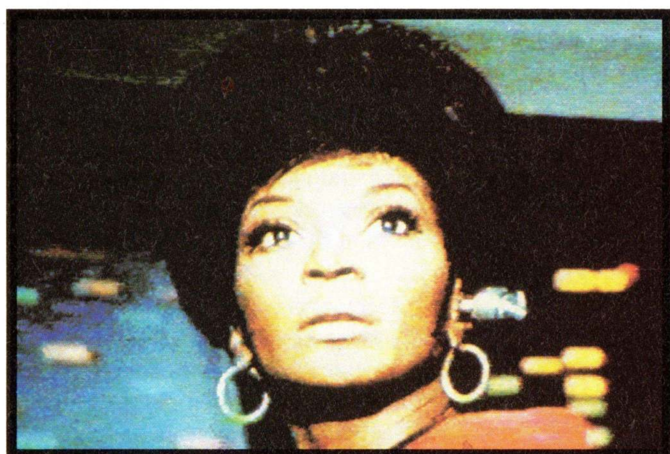
Das in diesem Test verwendete Zusatzgerät, der RGB-Splitter VD 3, nimmt dem Anwender das lästige und qualitativ nicht immer überzeugende Arbeiten mit den Farbscheiben ab, sofern man farbige Bilder digitalisieren möchte. Das Gerät, das mit einem eigenen Netzteil ausgeliefert wird, ist speziell für den VD 3-Digitizer entwickelt worden und verfügt über einen Farbvideoeingang und drei Ausgänge, an denen die Rot-, Grün- und Blausignale anliegen. Diese drei Ausgänge lassen sich direkt mit den Eingängen 2, 3 und 4 des VD 3-Digitizers verbinden, wodurch es möglich ist, eine Farbdigitalisierung in einem einzigen Durchgang vorzunehmen. Der RGB-Splitter läßt sich allerdings auch mit DigiView betreiben, wobei man allerdings zwischen jeder Digitalisierungsphase (für Rot, Grün und Blau) das Kabel, das zum Digitizer führt, umstöpseln muß. Die Firma Merkens bietet jedoch ein mit dem VD 3-Splitter technisch identisches Gerät an, das über nur einen einzigen Ausgang und einen Umschalttaster für die Farbanteile verfügt und sich somit speziell für DigiView anbietet. Da ein elektronischer RGB-Splitter Farbverschiebungen, wie sie bei der Anwendung von Farbscheiben entstehen, vermeidet, läßt sich mit einem solchen Gerät eine deutlich bessere Digitalisierungsqualität erreichen, was sich in der Anwendung mit beiden Digitizern bestätigte. Auch in Bezug auf die praktische Handhabung bringt ein Splitter gewaltige Vorteile mit sich, denn das Wechseln der Farbscheiben ist erstens umständlich und kann auch immer zu einem Verschieben der Kamera führen, wenn man einmal etwas unvorsichtig vorgeht, was dann zum Neubeginn der Digitalisierung zwingt. Für denjenigen, der viel und gut digitalisieren möchte, erweist sich ein solcher Splitter als optimale Unterstützung, wenn nicht sogar als ein Muß.

Geschwindigkeit ist Trumpf

In der Verarbeitungsgeschwindigkeit unterscheiden sich die beiden Geräte fast ebenso deutlich wie in den äußeren Abmessungen. Der VD 3 ist beispielsweise in der Lage, Schwarz-Weiß-Digitalisierungen in einer Rate von bis zu 10 Bildern pro Sekunde vorzunehmen. Dies gibt dem Anwender die Möglichkeit, sogar bewegte Objekte einzulesen. Auch erweitert dies den Anwendungsbereich des Digitizers um einige Varianten; so ist zum Beispiel eine Raumüberwachung, die auf Bewegungen reagiert, denkbar. Weiterhin verfügt der VD 3 über verschiedene Digitalisierungsmodi, wie zum Beispiel den Outline-Modus, der nur Konturen bzw. Farbübergänge darstellt, womit sich sehr interessante Effekte erzeugen lassen. Ein Beispiel hierfür ist bei gleichzeitiger Verwendung eines Genlocks die Umrandung sich bewegender Objekte mit den Outlines, was einem ambitionierten Videofreak mit Sicherheit einige neue Möglichkeiten bietet. Auch bei der Farbdigitalisierung kann die Verarbeitungsgeschwindigkeit des VD 3 überzeugen. Das Einlesen aller Farbsignale benötigt nur wenige Sekunden, und auch der Aufbau des Gesamtbildes, der danach erfolgt, läßt nicht lange auf sich warten. Bei DigiView sieht dies schon etwas anders aus. Zwar geht auch hier der Aufbau des Gesamtbildes recht schnell vonstatten, aber das Einlesen der einzelnen Bildsignale benötigt doch deutlich mehr Zeit, je nach Einstellung der Treibersoftware zwischen 15 Sekunden und einer Minute, wobei man im "Schnell-Modus" allerdings auch mit nachlassender Digitalisierungsqualität rechnen muß. Außerdem bietet DigiView nicht den Komfort des Einlesens in einem einzigen Durchgang, selbst bei Anwendung eines RGB-Splitters.



Eine Sängerin, digitalisiert mit Merkens VD 3, in Pal-Auflösung.



Saubere Farbwiedergabe auch bei wechselhaften Kontrasten: MerkensVD 3.



Die Lady von New Tek.

Praktische Probleme

Leider ist es beim Digitalisieren nicht damit getan, alle Geräte anzuschließen, die Kamera auszurichten und loszulegen. Um gute Bildqualität zu erreichen, muß man sich vor allem um optimale Lichtverhältnisse kümmern, was auch gute Kenntnisse der verwendeten Kamera (Weißabgleich!) erfordert, denn in dieser Hinsicht sind beide Digitizer recht verwöhnt. Vor allem DigiView neigt zu einer gewissen Farbüberbetonung, die sich in leichterem Maße allerdings auch beim VD 3 bemerkbar macht. Dies kann mit den Regelmöglichkeiten der Software bei beiden Geräten zwar teilweise ausgeglichen werden, gerade ein Rotstich der Digitalisierungen bleibt aber dennoch des öfteren bestehen. Dieser Effekt tritt allerdings kaum mehr bei indirektem, aber kräftigen Tageslicht auf. Beim VD 3 ist hierbei jedoch eine leichte Überbelichtung des Bildes auffällig, die mit dem Regelbereich der Einsteller bisweilen nicht mehr ausgeglichen werden kann. Hier wünscht

man sich einen etwas erweiterten Regelbereich. DigiView wiederum neigt in stärkerem Maße zu einer übertriebenen Farbsättigung, die sich zwar ausgleichen läßt, aber dieser Ausgleich geht dann zu Lasten des Kontrasts, wodurch die Bilder recht unkonturiert werden. All diese Effekte hängen aber, wie gesagt, sehr stark von den Lichtverhältnissen ab, was den Anwender zu einigen Versuchen zwingt, bis die optimale Ausleuchtung gefunden ist.
In der Bildqualität unter optimierten Bedingungen können beide Digitizer überzeugen, wobei der VD 3 einen leichten Vorsprung erringt, da DigiView auch dann noch - abhängig von den Farben des digitalisierten Motivs - bisweilen zu leichten Farbverfälschungen neigt, die allerdings nur sehr ambitionierte Anwender stören werden. Allerdings hat DigiView ein weiteres, sporadisch auftretendes Manko: Bisweilen werden Digitalisierungen mit vertikalen Störungen in Form von Schlieren versehen, was dann meistens zu einer erneuten Digitalisierung zwingt.



Unter optimalen Lichtbedingungen erhält man auch mit DigiView saubere Farbwiedergabe.

Die Software

Die Steuersoftware beider Geräte erweist sich als ausgereift und anwendungsfreundlich. Zum Merkens-Digitizer werden gleich zwei Treiber mitgeliefert, einer zum Scharz-Weiß- und einer zum Farbdigitalisieren. Beide bieten umfassende Regelmöglichkeiten, sowohl in Bezug auf Digitizer als auch auf den Amiga. Der VD 3 unterstützt sämtliche Bildauflösungen des Amiga und freundlicherweise auch volle PAL-Ausnutzung, was bei der DigiView-Software leider nicht implementiert wurde. Eine Möglichkeit, die die Merkens-Software nicht bietet, ist die nachträgliche Bearbeitung eines Bildes in Bezug auf Kontrast, Helligkeit, Farbanteile etc. Dies kann beim VD 3 nur vor einer Digitalisierung eingestellt werden. Allerdings digitalisiert der VD 3 ein Bild mit neuer Einstellung etwa in der gleichen Zeit, die DigiView für die Neuberechnung eines Bildes benötigt. Außer zur Bildverarbeitung und der Voreinstellung des Amiga bieten beide Digitizer noch einige andere Features, die weniger bedeutsam für die Digitalisierung sind, aber eine einfache Handhabung unterstützen. Selbstverständlich unterstützen beide Programme freien Diskettenzugriff, wobei die Merkens-Software mit komfortablen Zusatzfunktionen verwöhnt, wie dem Speichern von Bildern in Serie, wobei diesen ein vom Anwender festgelegter Name und eine vom Programm angehängte laufende Nummer zugewiesen wird. Alles in allem ist die Software beider Digitizer den Fähigkeiten der Geräte entsprechend und bietet kaum einen Anlaß zur Kritik.

Fazit

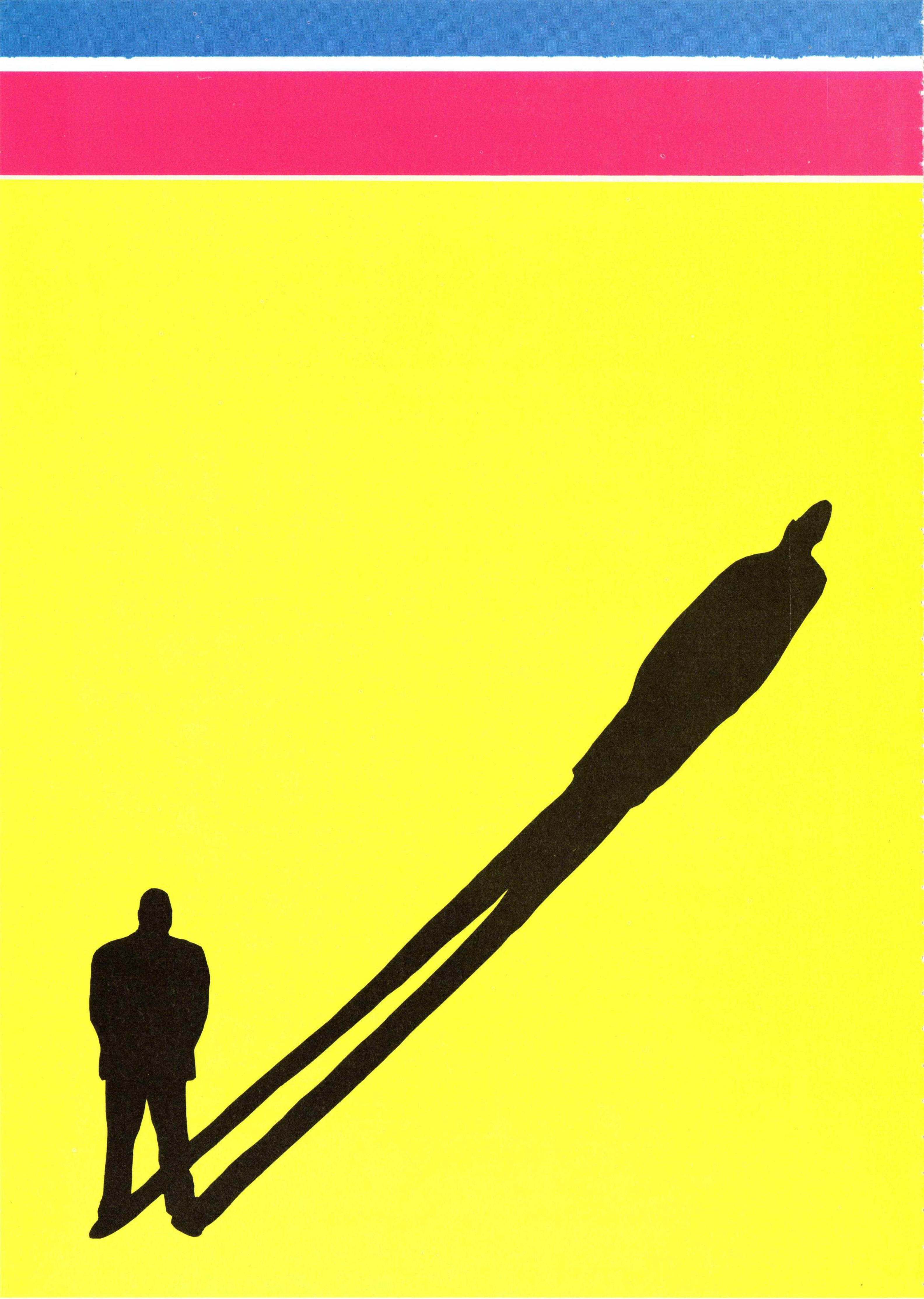
Die beiden vorgestellten Digitizer eignen sich für die verschiedensten Anwenderkreise. DigiView wird hierbei wohl eher den Kreis der Hobby-Digitalisierer ansprechen, für den er sich auch wegen seines geringen Preises anbietet. Höhere Qualität und größere Flexibilität bietet aber ohne Zweifel der Merkens-Digitizer, der allerdings auch um einiges teurer ist und sich damit für sehr ambitionierte Anwender anbietet. Man darf den großen Geschwindigkeitsvorteil des VD 3 auch nicht unterschätzen, denn wer einmal ein Dutzend Versuche unternommen hat, ein Motiv unter optimalen Lichtverhältnissen zu digitalisieren, der wird den Zeitgewinn zu schätzen wissen. Den Spaß an der Sache wird man aber ohne Zweifel mit beiden Geräten haben.

DigiView
Herst.: NewTek, Inc.
Preis: ca. 400.- DM
Vertrieb: Amiga-Fachhändler

VD 3 AMIGA
Herst.: Merkens EDV
Preis: 1798.- DM
Vertrieb: Merkens EDV
Fuchstanzstr. 6a

6231 Schwalbach

RGB-Splitter VD 3 AMIGA
Herst.: Merkens EDV
Preis: 898.- DM
Vertrieb: Merkens EDV
Fuchstanzstr. 6a
6231 Schwalbach



VON ANDREAS DIEBOLD

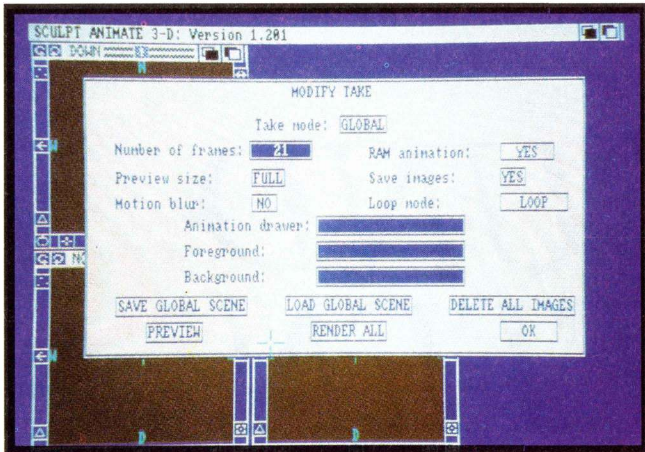
LICHT UND SCHATTEN

Ein lange erwartetes Programm ist nun endlich auf dem deutschen Markt erschienen. Nach Sculpt-3D, mit dem es möglich ist, fotorealistische Bilder im Ray-Tracing Verfahren zu erzeugen, soll nun Animate-3D aus dem gleichen Hause, Byte by Byte, ermöglichen, Bilder dieser Art zu einer Animation zusammenzufügen.

Denjenigen, die schon mit Sculpt-3D gearbeitet haben und die nicht unbeträchtlichen Rechenzeiten kennen, die zum Berechnen eines Bildes benötigt werden, wird sich zuerst die Frage stellen, ob es sich bei Animate-3D um ein völlig neues Programm handelt oder ob der Ray-Tracing-Teil unverändert (langsam) ist.

Die Antwort darauf ist sehr deutlich: Animate-3D ist kein eigenständiges Programm. Wer Sculpt-3D noch nicht sein eigen nennt und nun Animate-3D erwirbt, wird im ersten Moment der Benutzung eine Enttäuschung erleben. Zwar ist Animate-3D einst als ein zusätzliches Modul angekündigt worden, doch der Kaufpreis von 348,- DM läßt erwarten, daß man dafür ein lauffähiges Programm in die Hand bekommt. Dem ist leider nicht so. Die Entwickler scheuten keinen Aufwand, dem Benutzer die enge Verbundenheit von Sculpt-3D und Animate-3D zu demonstrieren. Vor der ersten Benutzung muß Animate-3D mittels eines mitgelieferten Programms mit Sculpt-3D "zusammengemerget" werden. Kurz gesagt: Nur wer schon Sculpt-3D erworben hat, kann die Investition von 348.- DM nutzen und mit Animate-3D arbeiten. Dies erhöht natürlich den Kaufpreis dieses Ray-Tracing Animations-Programms auf weit über 500,- DM, was aus der Werbung allerdings nicht so eindeutig hervorgeht. Ein Vorteil allerdings entsteht durch dieses System des "mergens". Man kann jegliche Version von Sculpt-3D mit Animate-3D verbinden. Außerdem verschwindet der Fehler, der beim Einladen von IFF-Bildern in PAL-Auflösung entstand. Die Rechengeschwindigkeit des Ray-Tracers bleibt jedoch unverändert. Nun aber zu dem, was einen nach dem "Mergen" erwartet. Auf den ersten Blick sieht alles nach Sculpt-3D aus: Der gewohnte Tri-View mit einer Vielzahl von Menüs. Rein äußerlich lassen sich nur einige neue Menüpunkte als Unterscheidungsmerkmal festmachen.

Das wichtigste Feature zum Erstellen einer Animation ist ein Take. Dieser Ausdruck stammt aus dem Wortschatz der Filmemacher und bedeutet soviel wie eine in sich geschlossene Aufnahmesequenz. Dementsprechend wird mit einem Take eine Animation global verwaltet. Es besteht aus zwei Windows, welche eine Vielzahl von Gadgets beinhalten. Es wäre natürlich müßig und sehr kompliziert, eine fließende Animation zu erzeugen, wenn man jedes Einzelbild separat konstruieren und berechnen müßte. Ein gutes Animationsprogramm hebt sich hier durch Hilfsmittel hervor, welche diese Aufgabe vereinfachen. Animate-3D bietet hier einiges. Die einfachste Möglichkeit ist das Arbeiten mit Keyframes. Wie gewohnt entwirft man in Sculpt eine Scene, welche das Start-Keyframe darstellt, also das erste Bild der Animation. Nun wird eine zweite veränderte Scene genutzt, um ein weiteres Keyframe darzustellen. Alle Objekte der Scene können hier ihre Position verändert haben. Im Take vereinbart man, wieviele Bilder zwischen diesen beiden Scenes berechnet werden sollen. Animate-3D interpoliert dann die Bewegung der Objekte, d.h., sie werden möglichst gleichmäßig von ihrer Start- zu ihrer Endposition bewegt. Man kann innerhalb eines Animationsablaufs auch mehrere Keyframes definieren, um den Bewegungsablauf besser kontrollieren zu können, denn besondere Wünsche kann dieses Verfahren natürlich nicht erahnen. Ebenso witzig wie interessant ist dabei natürlich auch die Möglichkeit der Metamorphose, also der Veränderung eines Objektes während der Animation. Doch uneingeschränkt ist dies nicht, da man die Grundstruktur eines Objektes nicht verändern darf, d.h. weder die Anzahl der Scheitelpunkte eines Objekts noch



Das Take dient zur Verwaltung einer Animation.

die Art der Verbindung dieser Punkte darf sich ändern. Doch auch so lassen sich schnell viele Effekte erzeugen, welche per Hand immens langwierig zu erzeugen wären.

Die zweite Methode, eine Animation zu beschreiben, ist ein Path, also ein Weg. So wie es der Name verspricht, wird ein solcher als ein einfaches Gebilde zusammenhängender Linien in das Tri-View gezeichnet. Menügesteuert wird eine solche Linie dann zum Path gemacht. Jeder Scheitelpunkt einer solchen Linie stellt ein Bild der Animation dar, genau an dem Punkt, an welchem sich das Objekt dann im Raum befindet. Es ist hierbei möglich, die Zeitabläufe zu kontrollieren, also den zeitlichen Abstand zweier aufeinanderfolgender Bilder, unabhängig von ihrer Entfernung, welche sich durch den Path ergibt.

Direkt zum Path gesellt sich ein weiteres Hilfsmittel, nämlich eine Hierarchieverwaltung. Um nämlich ein Objekt auf einem Path zu bewegen, wird dieses dem Path untergeordnet.

Doch dieses Hierarchiesystem läßt sich auch auf Objekte untereinander anwenden. Dies geschieht ähnlich dem File-System auf den Disks mit seinen Directories und Subdirectories in vielen Ebenen.

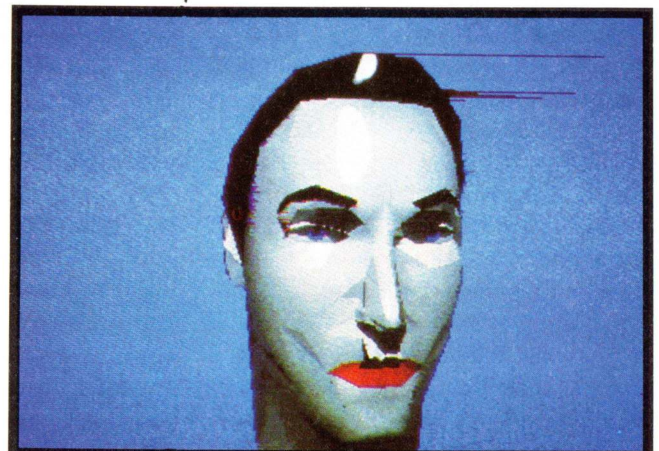
Mit dieser Hilfe kann ein Objekt, welches z.B. einem menschlichen Körper entspricht, verwaltet werden. So ist dann etwa der Arm dem Rumpf untergeordnet, dem Arm die Hand und der Hand wiederum die Finger. Mit dem schon angestellten Vergleich kann man den Rumpf als Hauptdirectory sehen, in der sich nun Subdirectories wie linker Arm, rechter Arm, linkes Bein etc. befinden. In den Arm-Subdirectories befindet sich nun wiederum die Directory Hand, in welcher sich dann als angenommener Endpunkt fünf Finger aufhalten. Der Vorteil der ganzen Aktion ist : Wenn man eine Directory bewegt, bewegt man ihren gesamten Inhalt mit. Trotzdem ist jeder Inhalt wiederum eine eigenständige Sache, welche auch separat bewegt werden kann. Bindet man auf diese Art mehrere Objekte zusammen, etwa die oben angenommenen, so wird, wenn der Rumpf bewegt wird, auch alles andere mitbewegt. Die Arme usw. bleiben also nicht irgendwo im Raum alleine und verlieren ihren notwendigen Zusammenhang mit dem Rumpf. Trotzdem bleiben sie eigenständige Objekte und können wie ein Arm etwa gehoben werden, ohne daß dabei wiederum der Rumpf bewegt wird. Man sollte sich allerdings seine Hierarchien schon vorher gut überlegen, da ein Einfügen oder Davorsetzten in der heirachischen Struktur nur durch einigen Aufwand möglich ist. Dazu muß man nämlich einen Teil löschen, dann das Gewünschte einfügen, um zuletzt wieder das Alte anzuhängen.

Besonders nett wird diese Sache noch durch die Möglichkeit, weitere Wege (Paths) in diese Strukturen einzubinden, um, z.B. wenn der Rumpf von a nach b bewegt wird, den Arm schwingen zu lassen. Und all dies läßt sich recht einfach auf grafischem Wege im Tri-View erstellen, ohne daß man dabei Unmengen von Koordinaten im Kopf oder sonstwo behalten muß.

Zum Thema Path bleibt noch anzumerken, daß die Lage des Objekt auf diesem Weg nicht dem Zufall überlassen bleibt. Ohne Zutun behält es die Lage in Bezug auf die Raumachsen, in der es konstruiert wurde, bei. Doch mit <MODIFY TUMBLE> läßt sich für jedes Bild die Lage des Objekts wiederum auf grafischem Wege edieren.

Eine weitere Neuigkeit sind Splines.

Das sind Linien, die durch mehrere Punkte beschrieben werden, diese aber nicht direkt, also gradlinig miteinander verbinden, sondern diese Verbindung in weichen Bogen beschreiben. Der Grad dieser Weichheit kann beliebig verändert werden. Ebenso kann die Linie auf ihren Definitionspunkten verzerrt oder geknickt werden. Es ist damit ein Leichtes, in sich verzogene Flächen zu erzeugen. Eine Spline kann auch zu einem Path gemacht werden, womit man eine weitere Möglichkeit hat, den Weg der Animation zu beeinflussen. Animate-3D läßt mehrere Wege offen, die Animation festzuhalten. Zum einen als komprimierte Bildfolge aus dem RAM, über ein Wiedergabeprogramm, das separat mitgeliefert wird und wohl weitergegeben

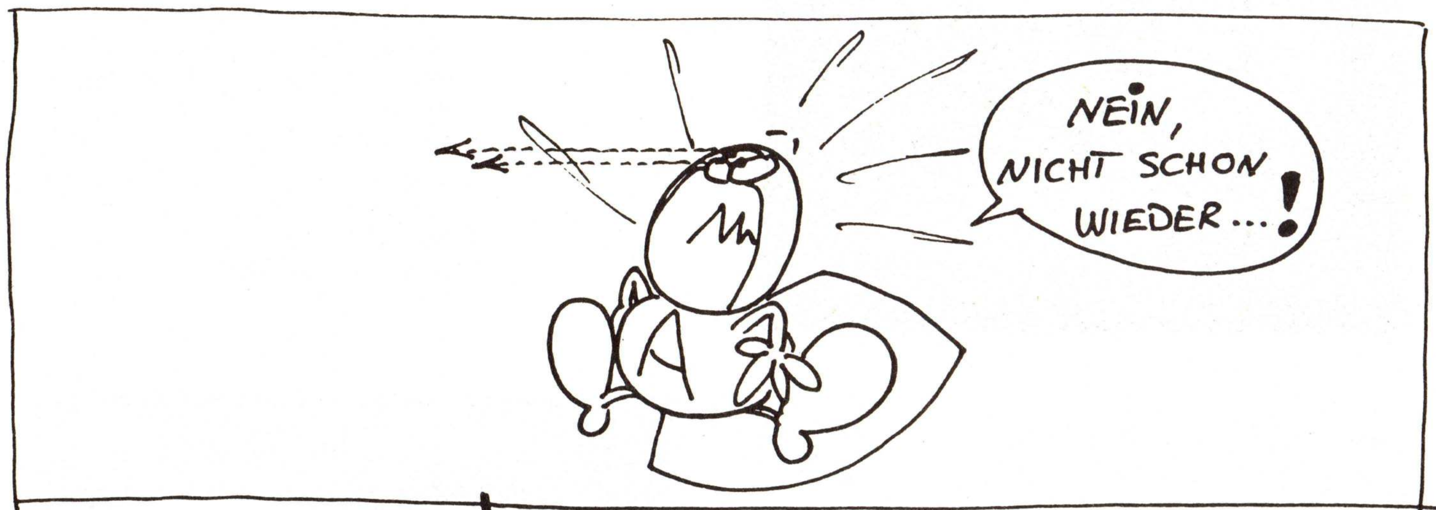


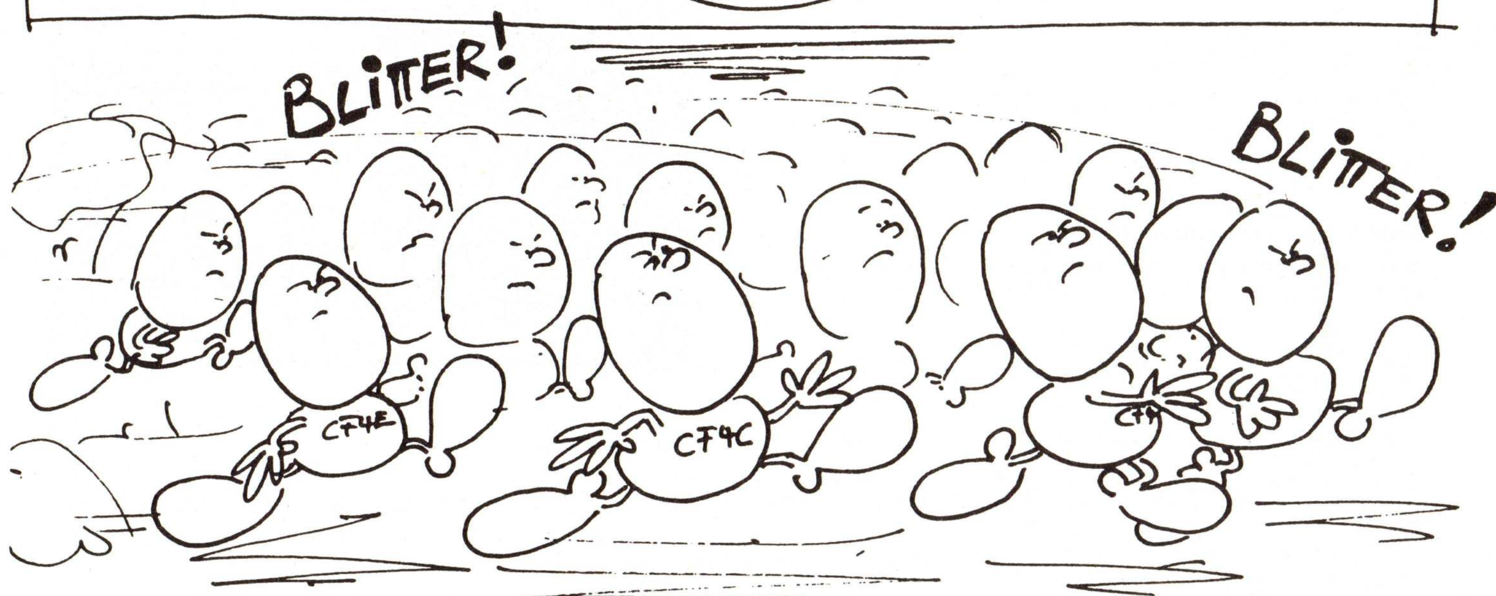
Bisweilen entstehen leider fehlerhafte Streifen im Bild.

AMIGA MAN

ABENTEUER IM AMIGA!

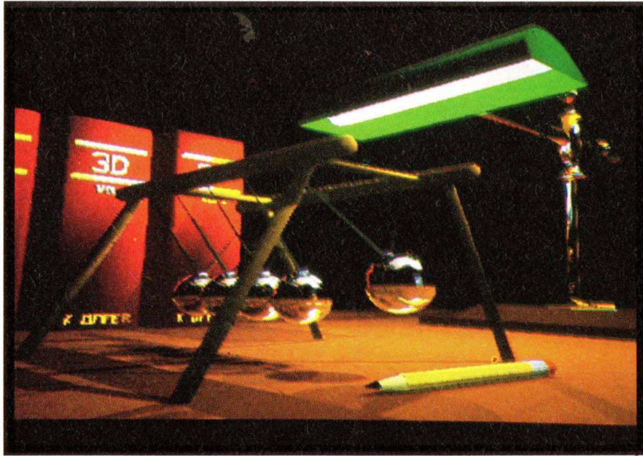








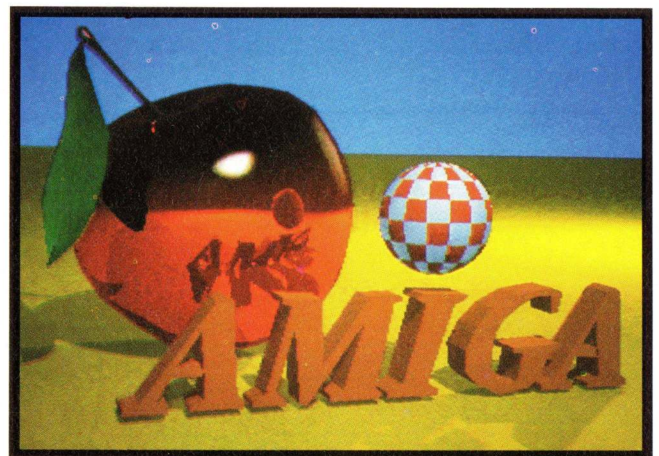
AB.87



Auch komplexe Arrangements sind kein Problem.

werden darf, oder direkt zu einem anderen Medium, etwa einem Videorecorder, der in der Lage ist, Einzelbilder aufzunehmen. Über die serielle Schnittstelle kann diesem dann ein Signal gegeben werden, wenn ein weiteres Bild fertig berechnet ist, so daß man nicht manuell darauf lauern muß. Diese Methode ist jedoch durch den enorm hohen Preis einzelbildaufnahmefähiger Videorecorder nur einem sehr begrenzten Anwenderkreis vorbehalten.

Bei der Anwendung der Methode der RAM-Animation tut sich in der vorliegenden Version ein kleines Problem auf. Laut Handbuch sollte man sich entscheiden können, ob jedes einzelne Bild als IFF-File erhalten bleibt oder nicht. Da die Komprimierung zum Film über den Weg der IFF-Files erfolgt, sollten, wenn man sich gegen die Speicherung aller Einzelbilder entscheidet, nie mehr als fünf Bilder auf einmal in den Massenspeichern vorhanden sein. Soweit das Handbuch. In der Praxis jedoch blieben immer alle Bilder auf den Laufwerken, bzw. in der RAM-Disk liegen. Bedenkt man nun, daß für eine Sekunde Animation 25 Bilder benötigt werden, so erfordert dies gigantische Kapazitäten an Massenspeichern, denn auch der eigentliche Animationsdatensatz nimmt recht schnell beachtliche Ausmaße ein. Einzig tröstend ist da noch die Option, daß man die Lage jedes einzelnen Bildes vorausbestimmen kann und so alle zur Verfügung stehenden Laufwerke nutzen kann. Dabei



Die bekannte Kugel-Demo, erstellt mit Animate 3D.

muß aber für jedes Bild eine eigene Angabe gemacht werden. Ebenfalls sehr störend fiel auf, daß während der Berechnung der Animation kein Abbruch möglich war. Wird man sich eines Fehlers erst zu spät bewußt, bleibt nur noch sehr lange warten oder ein Reset. Man sollte allerdings hoffen können, daß dieser Fehler schnell beseitigt wird, da er doch recht hinderlich ist. Auch an anderen Stellen läuft noch nicht alles hundertprozentig, und Benutzerfehler werden mitunter ein Mekka für den Guru. Trotzdem, die Möglichkeiten sprechen für sich, und nach etwas Eingewöhnung kann man einige tolle Sachen machen, wenn man bereit ist, für einige Zeit auf anderweitige Nutzung des Rechners zu verzichten. Außer den bekannt langen Rechenzeiten des Ray-Tracers geht auch die Komprimierung der Bilder zur Animation verblüffend langsam vonstatten.

Die Komprimierung von einem Bild zum nächsten kann schon mal eine viertel bis halbe Stunde dauern. Bei einer Animation über fünfzig Bilder vergeht da dann schnell nochmal ein weiterer Tag, bis man sich seiner Mühen erfreuen darf. Für intensive Nutzung sind also am besten wenigstens zwei MegaByte und eine Turbokarte angesagt. Man darf jedoch nicht vergessen, daß auch Animationen im Paint-Modus, also ohne Ray-Tracing, möglich sind.

Das geht viel schneller vonstatten und ergibt auch eindrucksvolle Ergebnisse, die etwa Videoscape 3D in Bedienung und Qualität (freie Farbwahl) überlegen sind.

Fazit: Wo viel Licht ist, da gibt es auch viel Schatten, wozu sich ja auch noch der hohe Preis gesellt. Ob die Methode des "Mergens" (oder sollte man es vielleicht auch "Ein Programm zum Preis von zweien" nennen?) viele Freunde finden wird, bleibt zu bezweifeln. Jedoch läßt es sich nicht von der Hand weisen, daß diese Software enorm leistungsfähig und dem Amiga würdig ist, ihn also in Bereiche vorstoßen läßt, die noch vor nicht allzulanger Zeit für dieses Geld als undenkbar erschienen.

GOLEM

Wir liefern im 3-Tage-Rhythmus

KUPKE



02 31/81 83 25-27
Telefax 02 31/81 74 29
D-4600 Dortmund 1
Burgweg 52a

1
2
3
4
5
6
7

1 Golem Drive 3,5

NEC 1036a mit heller Frontblende ● Amiga-farbenes Metallgehäuse ● Abschalter ● Busdurchführung bis DF3 ● PC-Karten und Sidecar kompatibel **!!! neu !!! jedes Drive mit Trackdisplay** zur aktuellen Spur- und Kopfanzeige

mit Display DM **379.-**
ohne Display DM **369.-**

2 Golem Drive 5.25

5,25 Zoll Laufwerk mit heller Frontblende ● Amiga-farbenes Metallgehäuse ● Abschalter ● Busdurchführung bis DF3 ● 40/80 Track Umschalter Amiga und MS-Dos kompatibel **!!! neu !!! Drive mit Trackdisplay wie Golem 3,5**

mit Display DM **449.-**
ohne Display DM **439.-**

3 Golem Drive 3,5 intern

modifiziertes NEC 1036a mit heller Blende ● Staubschutzklappe zum Einbau in den A 2000 incl. Einbausatz und Einbauanleitung

DM **269.-**

4 Golem Ram Box 1000

2 MB Erweiterung ansteckbar ● in Amiga-farbenem Metallgehäuse ● Abschalter ● Busdurchführung ● auto konfigurierend ● Betriebskontrollanzeige durch LED ● erweitert den Grundspeicher auf 2,5 Megabyte

DM **998.-**

Technische Änderungen vorbehalten

5 Golem 500

Ram Erweiterung speziell für den Amiga 500 ● technische Einzelheiten wie Golem Ram Box 1000 ● beide Erweiterungen ohne Wait States

DM **998.-**

6 Kickstart / Uhr Modul

"Bitte Workbench einlegen", so meldet sich Ihr Amiga 1000 mit dem Kickstart Eprom Modul ● Ansteckbar am Systembus ● Amiga-farbenes Metallgehäuse ● durchgeführter Systembus ● abschaltbar sodaß andere Kickstart Versionen wieder gebootet werden können.

DM **199.-**

Amiga 2000 u. 500 kompatibeles, externes Uhrenmodul ansteckbar am Systembus ● Software, die die 2000/500 Uhr anspricht, benutzt auch die Golem Clock für den A 1000

DM **149.-**

Uhr und Kickstart in einem Gehäuse DM **299.-**

7 Golem Sound

Audio Digitizer der Spitzenklasse, kompatibel zu aller gängigen Software mit DIN- und Cinch-Anschluß auch für Micro Anschluß geeignet ● optische Aussteuerung über ein LED Display ● STEREO ● Wandlungsfähig ● 1MHz getaktet bietet der Golem Sound unglaubliche Sample Qualität.

Mono DM **139.-**
Stereo DM **189.-**
Software zum Golem Sound, stereofähig DM **29.-**

Entwurf Firma DataCom, Dortmund · Realisierung, Plum, 5014 Kerpen-Sindorf, Tel. 0 22 73/5 30 91

VON WOLF DIETRICH

UNTER DRUCK GESETZT

DREI DRUCKER, DIE FARBE AUF´S PAPIER

Wer am Rechner viel mit grafischen Programmen arbeitet, dem wird es sicher öfter einmal passieren, daß er einen Ausdruck seiner Leistungen zu Papier bringen möchte. Hier ist ein normaler Drucker allerdings nicht gerade das Gelbe vom Ei, denn ohne die Farben wirken oftmals die besten Ausdrucke nicht besonders. In diesem Fall ist die Anschaffung eines Farbdruckers durchaus sinnvoll, und wenn dieser dann auch noch hochwertige Textausdrucke beherrscht, hat man gleich zwei Fliegen mit einer Klappe geschlagen.

Drei Drucker, die dies alles vermögen, stehen sich in diesem Test gegenüber: der *Commodore MPS 1500C*, der *Xerox Diablo C 150* und der *PaintMaster* von *Calcomp*. Ein direkter Vergleich zwischen diesen Geräten soll allerdings aus zwei Gründen nicht gezogen werden: erstens arbeiten alle drei Drucker mit einem anderen Prinzip, und zweitens liegen sie preislich sehr weit auseinander. Dies gilt im Vergleich zu den zwei anderen Geräten vor allem für den Calcomp, der eher ein Beispiel für das heutzutage technisch Machbare darstellt.

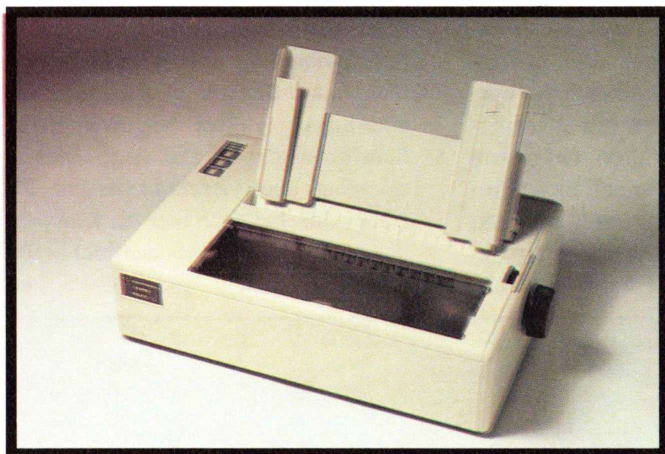
Der *MPS 1500C* ist ein Vertreter der Gattung der Matrixdrucker und arbeitet mit 9 Nadeln. Er ist mit Abstand das preisgünstigste Modell im Test, und mit einem Preis von 798.- DM liegt er sicherlich an der Untergrenze dessen, was man für einen Farbdrucker allgemein auf den Ladentisch blättern muß. Schon etwas mehr muß man für den *Diablo C 150* hinlegen, denn dieses Gerät, übrigens ein Tintenstrahlfarbdrucker, schlägt mit 1998.- DM zu Buche. Dies sind jedoch alles keine Summen im Vergleich zum *Calcomp PaintMaster*. Wer dieses Gerät, das nach dem Thermotransferprinzip arbeitet, sein eigen nennen möchte, sollte über ein größeres Bankkonto verfügen und/oder einen professionellen Einsatzbereich haben, denn es kostet ca. 12500.- DM.

Alltagsbetrieb

Wenn man viel zu drucken hat, so ist die Bedienungs- und Einstellfreundlichkeit eines Druckers ein nicht unwichtiges Kriterium. Bereits hierbei überzeugt der *Calcomp* auf Anhieb, denn bei diesem Gerät hat der Anwender nicht viel mehr zu tun, als den Druckertreiber zu laden, Papier einzulegen und mit dem Ausdrucken zu beginnen. Außer einem Einschalter verfügt der *PaintMaster* nur noch über zwei Druckknöpfe an der Vorderseite, die zum Papiereinzug und -auswurf dienen, was der Drucker im normalen Betrieb allerdings auch automatisch erledigt. Um Einstellungen eines Zeichensatzes oder verschiedener Modi muß sich der Anwender bei diesem Gerät nicht kümmern, denn das wird vom Druckertreiber vorgenommen. So erwies es sich nach dem Laden desselben als problemlos, deutsche Sonderzeichen, Fett- und Kursivschriften und auch Unterstreichungen auszugeben. Dies gelingt beim*Diablo C150* leider nicht; dieses Gerät ignorierte alle verschiedenen Schriftarten einfach. Die Bedienung hingegen erweist sich als fast so unkompliziert wie die des Calcomp, denn an der Vorderseite findet man nur zwei Schalter für Reset bzw. Pause und Papiervorschub. An der Rückseite des Geräts gibt es noch Schalter für Selbsttest und Düsenreinigung sowie diverse Dipschalter für Grundeinstellungen, von denen eigentlich nur zwei (für die Wahl zwischen bi- und unidirektionalem Ausdruck und Aktivierung des automatischen Line Feeds) von größerer Bedeutung sind. Auch der Commo-

dore weist nur drei Schalter (Line Feed, Form Feed und Local bzw. Online) auf, aber bei diesem Gerät gibt es dennoch deutlich mehr Einstellmöglichkeiten. Diese werden allerdings nicht über Schalter, sondern im Dialogbetrieb vorgenommen. Hierbei druckt der *1500C* die jeweilige "Anfrage" aus, worauf man über die drei Schalter Veränderungen vornehmen kann.

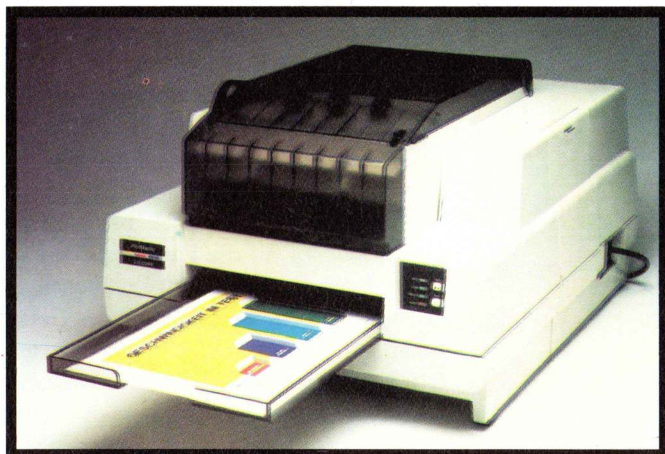
Dieses Prinzip erweist sich allerdings als etwas umständlich, denn bisweilen muß man sich durch den ganzen Dialog arbeiten, wenn man nur eine Einstellung ändern will. Dafür kann sich der Besitzer eines *1500C* über den mitgelieferten (unidirektionalen) Traktor für Endlospapier freuen. Auch ist dieses Gerät problemlos in der Lage, Fettschrift und Unterstreichungen auszugeben. Kursivschrift verweigerte er allerdings auch. Hierbei ist dazuzusagen, daß bei allen drei Druckern lediglich der dazugehörige Treiber geladen und dann der Druckvorgang aktiviert wurde, denn es ging darum, verschiedene Schriftarten ohne große Justage auszudrucken, wie das eben auch im Alltagsbetrieb funktionieren sollte. Der *Commodore* und der *Calcomp* bedürfen im normalen Betrieb keiner größeren Wartung; man muß lediglich ab und an das Farbband wechseln und dafür Sorge tragen, daß Papier eingelegt ist. Etwas mehr Arbeit hat man in dieser Hinsicht prinzipbedingt mit dem *Diablo*, denn mit dem Auswechseln der Tintenpatronen ist es nicht immer getan. Zum ersten muß man darauf achten, daß dieses Gerät absolut gerade steht, wozu es über eine eingebaute kleine Wasserwage verfügt, denn sonst bekommt man Probleme mit dem Tintenfluß. Auch bedarf der *C 150* regelmäßiger Reinigung, vor allem, wenn das Gerät mehrere Tage nicht in Betrieb war, da sonst die Qualität der Ausdrucke deutlich nachläßt. Im Lieferumfang sind dementsprechend einige Pflegegerätschaften inbegriffen. Der Reinigungsvorgang ist zwar nicht allzu arbeitsaufwendig, doch darf man sich danach im allgemeinen (wie auch beim Wechseln der Tintenpatronen) über bunte Finger freuen. Die Druckkosten halten sich bei *Commodore* und *Diablo* in Grenzen, denn weder das Farbband des einen noch die Tintenpatronen des anderen sind allzu teuer. Über den Preis des Farbbands für den *Calcomp* lagen leider keine Informationen vor, doch man muß davon ausgehen, daß er sich schon in anderen Dimensionen bewegt als der Preis für ein Matrixdruckerfarbband. Auch benötigt man für dieses Gerät Spezialpapier, welches nicht gerade billig ist. Allerdings ist auch beim *Diablo* die Verwendung von Spezialpapier ratsam, denn damit läßt sich die Qualität der Ausdrucke steigern. Sowohl der *Diablo* als auch der *Calcomp* sind übrigens in der Lage, für Overhead-Vorlagen Folien zu bedrucken.



Der MPS 1500 C von Commodore



Der Diablo C150 von Xerox.



Der PaintMaster von Calcomp.

Druckqualität

In der Qualität der Ausdrucke bestehen bei diesen drei Geräten deutliche Unterschiede, die schon von den Preisen her zu erwarten sind. So kommt der

Dies ist ein Textausdruck
mit dem Commodore 1500 C.
Wie sieht es hier mit den
deutschen Sonderzeichen aus:
ä ö ü Ä Ö Ü ß
Wunderbar. **Nun wird's fett,
und nun unterstrichen und fett.**
Leider gibt's mit den kursi-
ven Zeichen Probleme. Als
Druckertreiber wurde hier
ein IBM-Kompatibler verwen-
det, der MPS 1500 C emuliert
allerdings auch den Epson
(JX-80).

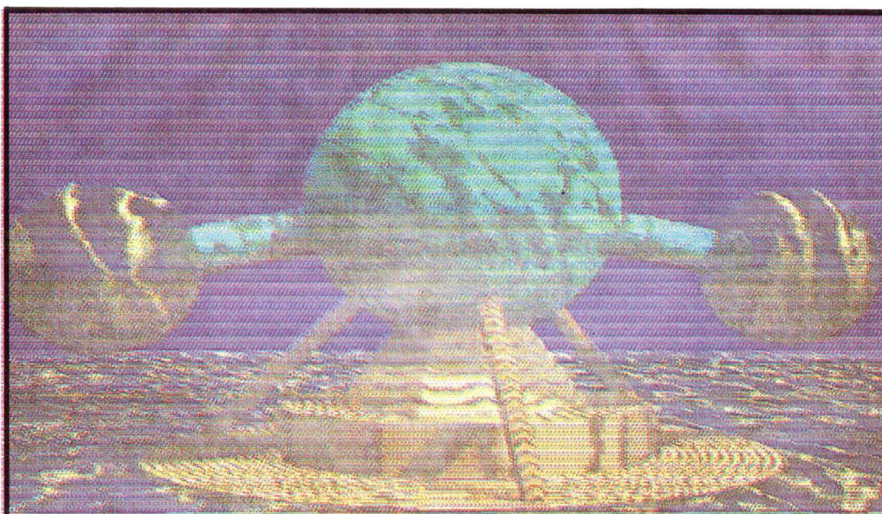
**Dies ist ein Textausdruck
mit dem Diablo C150 von
XEROX. DerTest mit den
deutschen Sonderzeichen:
ä ö ü Ä Ö Ü ß
Das macht keine Probleme.
Leider druckt der C150 we-
der in Kursiv- noch in
Fettschrift, und auch Un-
terstreichungen wurden
nicht ausgegeben.
Der passende Druckertreiber
für den C150 sollte auf je-
der Workbench zu finden
sein.**

Dies ist ein Textausdruck
mit dem PaintMaster von
CALCOMP. *Hier sieht man
Kursivschrift,* **und hier
Fettdruck.** ***Auch kursiv
und fett ist kein Pro-
blem.*** Das gleiche gilt
für Unterstreichungen,
mit *jeder Schriftart.* Nun
mal sehen. wie es um die
deutschen Sonderzeichen
steht:
ä ö ü Ä Ö Ü ß
All das funktioniert wun-
derbar, wenn man auf der
Workbench den Epson-Druk-
kertreiber anwählt. Das
Schriftbild gibt keinen
Anlaß zur Kritik.

MPS 1500 C mit helleren Bildern deutlich besser zurecht als mit sehr farbintensiven Ausdrucken, wo er öfters etwas zum Verschwimmen neigt. Auch zeigen sich die matrixdruckertypischen horizontalen Streifen recht deutlich, was ebenfalls bei dunklen Bildern am deutlichsten auffällt. Dennoch kann die Druckqualität durchaus befriedigen. Das Problem mit den Streifen kennt der *Diablo C 150* nicht. DiesesGerät kommt sehr gut mit den verschiedenartigsten Bildern zurecht, wobei man in bestimmten Fällen lediglich leichte Abstriche im Kontrast hinnehmen muß. Dafür sind Farben deutlich satter und voller, und auch die Punkte, aus denen das Bild aufgebaut ist, sind kleiner als beim *MPS 1500 C*. Allerdings hat der *Diablo* auch Probleme mit Pastelltönen, die er bisweilen ein wenig zu farbintensiv wiedergibt. Dieses Problem tritt allerdings nur in selteneren Fällen auf. Dem *Calcomp* hingegen sind all diese Probleme unbekannt, denn er glänzt mit einer bisher nicht gesehenen Druckqualität. Egal, welche Farben, welche Kontraste oder welche Bildgröße, der *PaintMaster* bringt die Bilder brilliant zu Papier. Dies geht so weit, daß man Ausdrucke von hervorragenden Digitalisierungen im DIN A4-Format bei einem Abstand von zwei Metern fast für Fotos halten kann. Diese Qualität liegt zum einen in der hohen Punktdichte, zum anderen aber auch in der Fähigkeit des *Calcomp*, beliebige Farben sauber "anzumischen", begründet. In der Qualität der Textausdrucke ist die Reihenfolge ähnlich, wenn auch der *Commodore* hier mit dem Diablo mehr oder weniger gleich zieht, was allerdings am etwas ungewöhnlichen Schriftbild des *C 150* liegt. Dies ist allerdings auch Geschmackssache. Die Nase vorn hat auch hier der *Calcomp*, der ein sehr sauberes Schriftbild liefert, das eigentlich nur noch von Laserdruckern übertroffen wird. Einen Vorteil gegenüber den beiden anderen Geräten hat der *Calcomp*, neben der hohen Druckauflösung, auch dadurch, daß die Buchstaben einfach absolut schwarz gedruckt werden.



Kontrastprobleme: MPS 1500 C



Matrixdruckertypische Streifen: MPS 1500 C

Bei den Druckzeiten wird der *Calcomp* allerdings zum Schlußlicht. Mit knapp 9 Minuten benötigt er die längste Zeit von allen Testgeräten für einen beliebigen Farbausdruck mit mittlerer Einstellung für die Farbintensität (im Preferences-Menü der Workbench). Man muß zur Verteidigung des Geräts allerdings sagen, daß dies einzig und allein dem Druckertreiber anzulasten ist, denn der *Calcomp* benötigt selbst nur eine Minute für einen Farb-

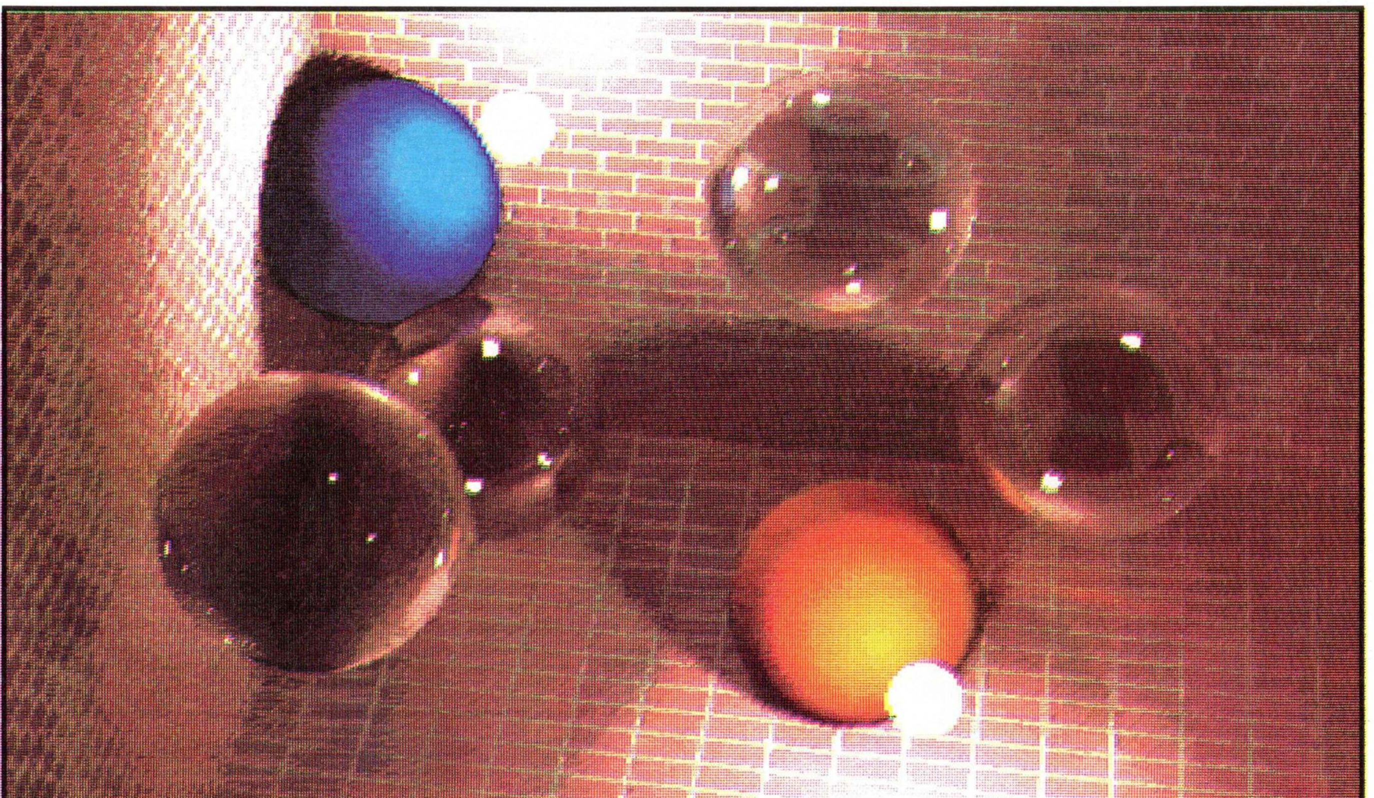


Sanfte Farben: Diablo C 150

ausdruck. So kann man beobachten, daß der Drucker innerhalb der gemessenen 9 Minuten die meiste Zeit wartet (das Wait-Lämpchen blinkt). Und entzieht man dem Drucker das Papier und läßt erst einmal den Treiber die Daten übertragen, so wird nach dem Einlegen des Papiers dieses eingezogen und tatsächlich innerhalb einer Minute bunt bedruckt wieder ausgegeben. Den zweiten Platz nimmt in der Geschwindigkeitsbewertung der *MPS 1500 C* ein, der für einen Farbausdruck ca. 7 1/2 Minuten benötigte. Der *Diablo* setzt sich von den beiden anderen mit knapp 5 Minuten Druckzeit für ein Farbbild noch einmal deutlich ab. Auch in punkto Geräuschentwicklung fällt der *Diablo* am angenehmsten auf, denn man hört bei ihm, abgesehen von einem fast unmerklichen Summen und einem leichten Anschlagsgeräusch des Druckkopfs am Zeilenende, nichts weiter. Auch der *Calcomp* arbeitet angenehm leise, eigentlich sogar unhörbar, macht aber durch deutliches Lüfterrauschen auf sich aufmerksam, was sich aber noch in erträglichen Grenzen hält. Etwas anders verhält sich dies beim *Commodore*, dessen Nadelarbeit zu einigen Teilen in akustische Energie umgesetzt wird. Dies ist einerseits recht laut und bewegt sich andererseits auch noch in einem etwas schrillen Frequenzbereich, so daß Dauerdrucken mit diesem Gerät bisweilen leicht nervenbelastend ausfällt.

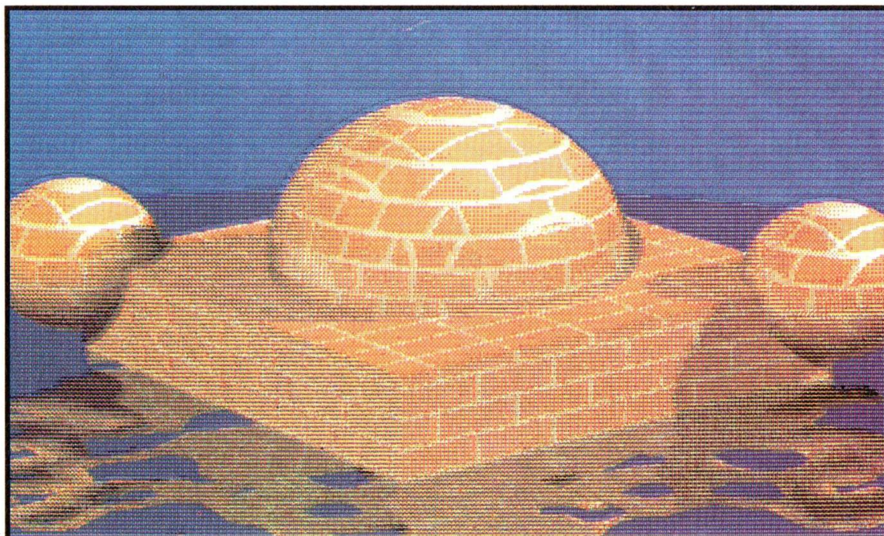
Fazit

Alle drei vorgestellten Geräte bieten sich für unterschiedliche Einsatzbereiche an. Der *Commodore* erweist sich als billiger Allround-Drucker, für akzeptable Textausdrucke in NLQ und gelegentliche Grafikausdrucke gleichermaßen geeignet. Wer allerdings viel Wert auf den Grafikausdruck legt, dem wird die Qualität des *MPS 1500 C* nicht ausreichend sein, und so bietet sich der *Diablo* an, der zwar etwas mehr Wartung verlangt, dafür aber mit deutlich besseren Farbausdrucken aufwarten kann. Dieser Qualitätssprung



Sanfte Farben kein Problem: Calcomp PaintMaster

muß natürlich mit einem höheren Preis erkauft werden. Der *Calcomp* andererseits ist kein Gerät für den Heimanwender, sondern bringt zu einem entsprechenden Preis professionelle Druckqualität in Farbe, die keines der anderen Geräte auch nur annähernd erreichen kann. Er dürfte damit für geschäflichen Einsatz prädestiniert sein. Man muß wohl auch berücksichtigen, daß man für den extrem hohen Preis einiges geboten bekommt, wie z.B. 1,5 Megabyte Druckpuffer (was wohl sonst nur Laserdruckern vorbehalten ist) und Kompatibilität mit verschiedenen professionellen Grafikstandarddatenformaten. Die einzige Kritik an diesem Gerät beschränkt sich unter diesen Gesichtspunkten auf den langsamen und unprofessionell gestalteten Druckertreiber.

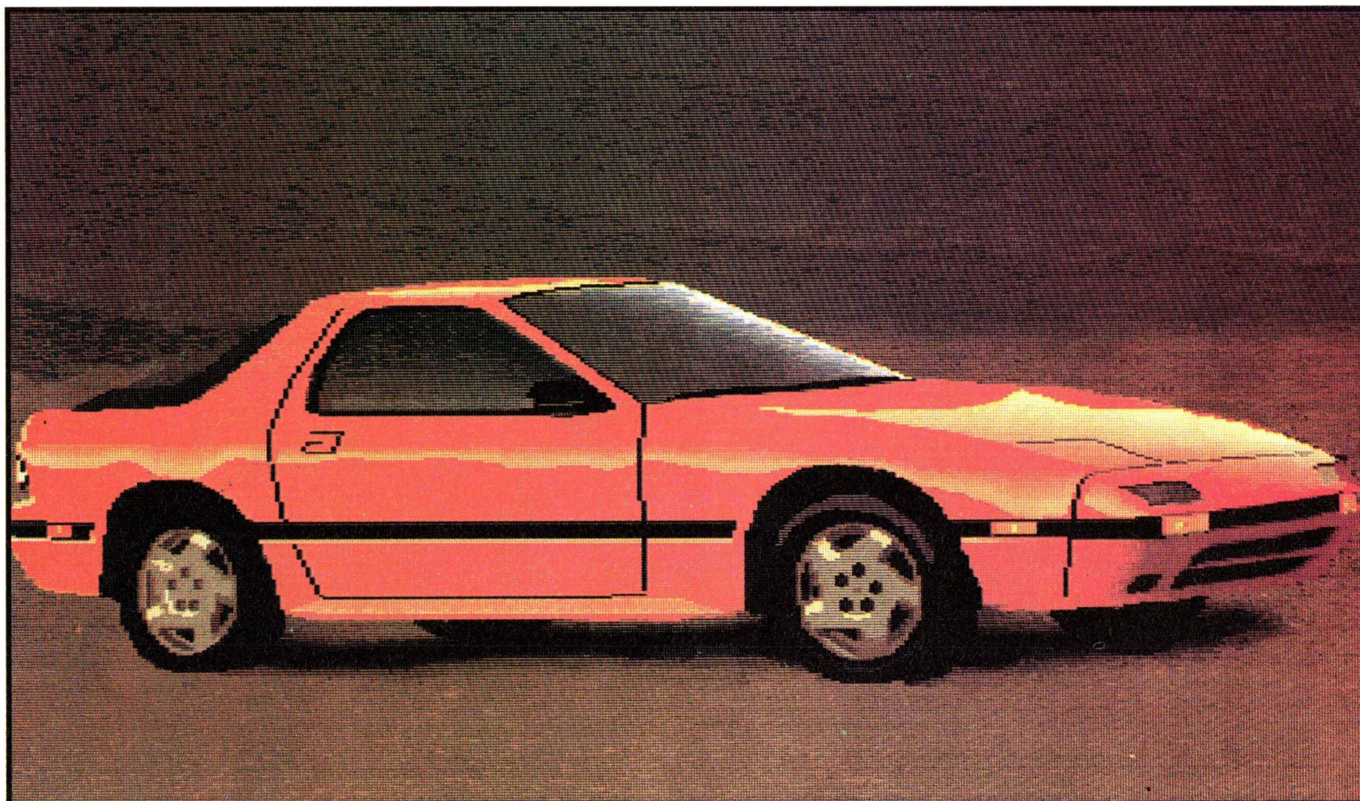


Volle Farben ohne Streifen: Diablo C 150

Commodore MPS 1500 C 9-Nadel-Matrixdrucker, Epson JX80 kompatibel
Preis: 798.- DM
Bezugsquelle: DTM, Wiesbaden, 06121/560084

Xerox Diablo C150 Tintenstrahlfarbdrucker
Preis: 1998.- DM
Bezugsquelle: DTM, Wiesbaden, 06121/560084

Calcomp PaintMaster Thermotransferdrucker, Epson FX80-kompatibel
Preis: ca. 12500.- DM
Bezugsquelle: PDC, Bad Homburg, 06172/24748



Red as red can be: Calcomp PaintMaster

VON MARKUS NERDING

BÜCHER FÜR ANGEHENDE ZEICHENKÜNSTLER

Walter Friedhuber Computermalschule Trickfilmzeichnen

In diesem Teil der "Computermalschule" widmet sich Walter Friedhuber ausschließlich der Erstellung von Trickfilmen. Voraussetzung sind dabei in erster Linie ein Malprogramm und das Animationsprogramm DeLuxe Video. Schritt für Schritt werden im Buch die einzelnen Phasen erklärt, die zur Erstellung eines kompletten Trickfilms notwendig sind. Dies beginnt bei der Idee, geht über die ersten Skizzen der Bewegungsabläufe bis zur fertigen Animation. Dabei werden die Grundlagen des Animationszeichnens besonders ausführlich erläutert: Ein Skelettmodell wird nur aus stilisierenden Strichen erstellt und in Bewegung versetzt. Danach kommen zu diesem Modell noch zylindrische Elemente hinzu, die erste Eindrücke der Figuren hinterlassen. Erst wenn mit diesem Modell die Animation entworfen wurde, beginnt man, die Figuren in ihrer endgültigen Form zu zeichnen und mit Farben zu versehen.

Das Buch beschreibt die Animationstechniken und auch deren Grundlagen sehr anschaulich und ausführlich. Zahlreiche Handskizzen und Hardcopies (leider nur mit sehr magerer Qualität!) illustrieren den Text und tragen dadurch sehr zum Verständnis bei. Das Schönste an diesem Buch sind jedoch die beiden mitgelieferten Disketten. Sie enthalten fast alle Animationssequenzen, die im Buch beschrieben werden. Zusätzlich sind auch noch die Hintergrundbilder, die Figuren und die notwendigen Brushes vorhanden, so daß man direkt mit der Anleitung des Buches ein Video erstellen kann, ohne sich erst lange mit dem Erstellen der Grundelemente beschäftigen zu müssen. Neben den im Buch vorgestellten Zeichentricksequenzen lassen sich mit den vielen Hintergrund- und Animationsobjekten auch sehr leicht eigene Szenenabläufe erstellen, in denen vorhandene und selbsterstellte Objekte vorkommen.

Das Arbeiten mit dem Buch macht viel Spaß. Besonders durch die Objekte und fertigen Filme, die auf den Disketten mitgeliefert werden, fällt es auch dem Einsteiger in dieses interessante Thema leicht, schrittzuhalten. Wer sich zuerst die Videos anschaut (und wer wird das nicht tun!), der wird sich gleich darauf auf das

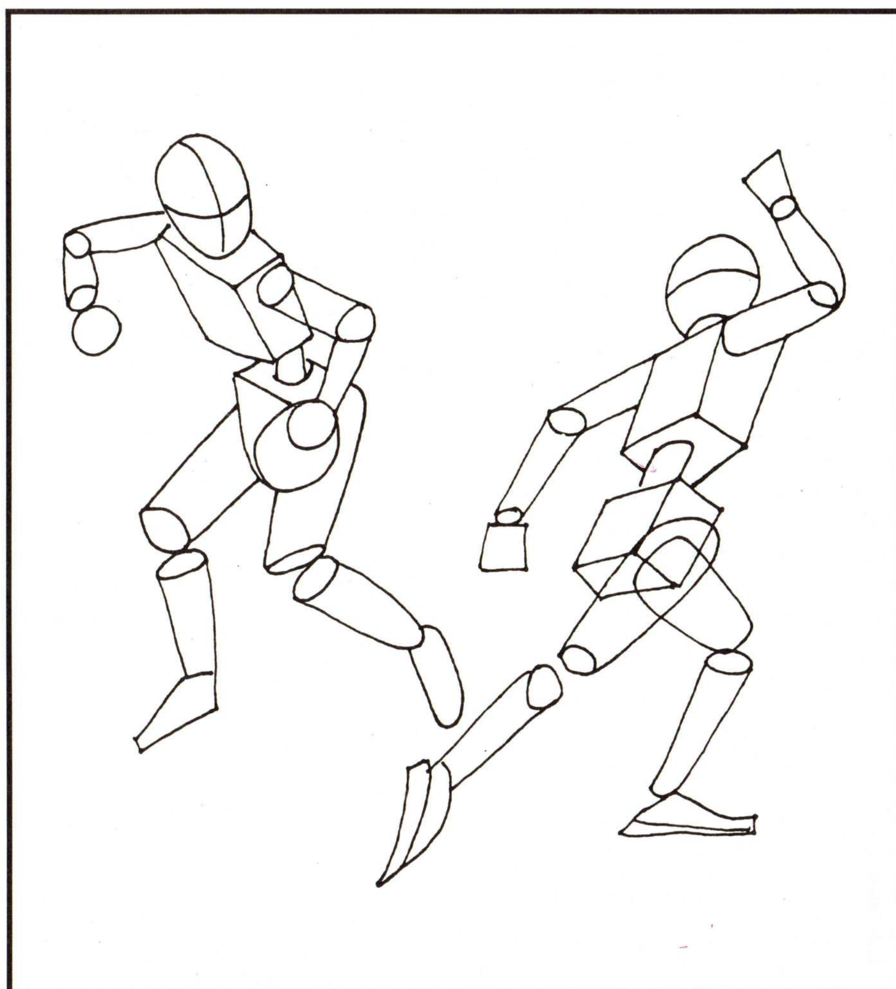
Buch stürzen, um die Techniken und Tricks kennenzulernen.

Bezugsquelle:
Verlag Gabriele Lechner
Preis: DM 59.- (incl. 2 Disketten)

Walter Friedhuber & Jimmy Stepanoff

Professionelles Arbeiten mit Deluxe Paint II

Das Buch umfaßt gut 600 Seiten, und was darin geboten wird, kann sich sehen lassen. Nach einer kurzen Einführung in die Hard- und Software des AMIGA werden ausführlich die Grundlagen des Zeichnens erklärt: Linien, Portrait, menschliche Körper, Licht und Schatten, Farben, Perspektive und Bildkompositionen. Schon hier zeigt sich die Vorliebe der Illustratoren für das "Fantasy"-Genre, das viel Spielraum für kreative Gestaltungen und Zeichnungen bietet. Weitere Grundlagen findet man im Kapitel "Im Computer-Atelier", in dem sich der Autor noch einmal ausführlich den Themen Portrait- und Aktzeichnen sowie Comics widmet. Ausgerüstet mit den Grundlageninformationen nähert man sich dem dritten Teil des Buches, in dem auf



Bewegungsstudien.



Auf dem Screen sieht sie besser aus als auf dem Ausdruck.

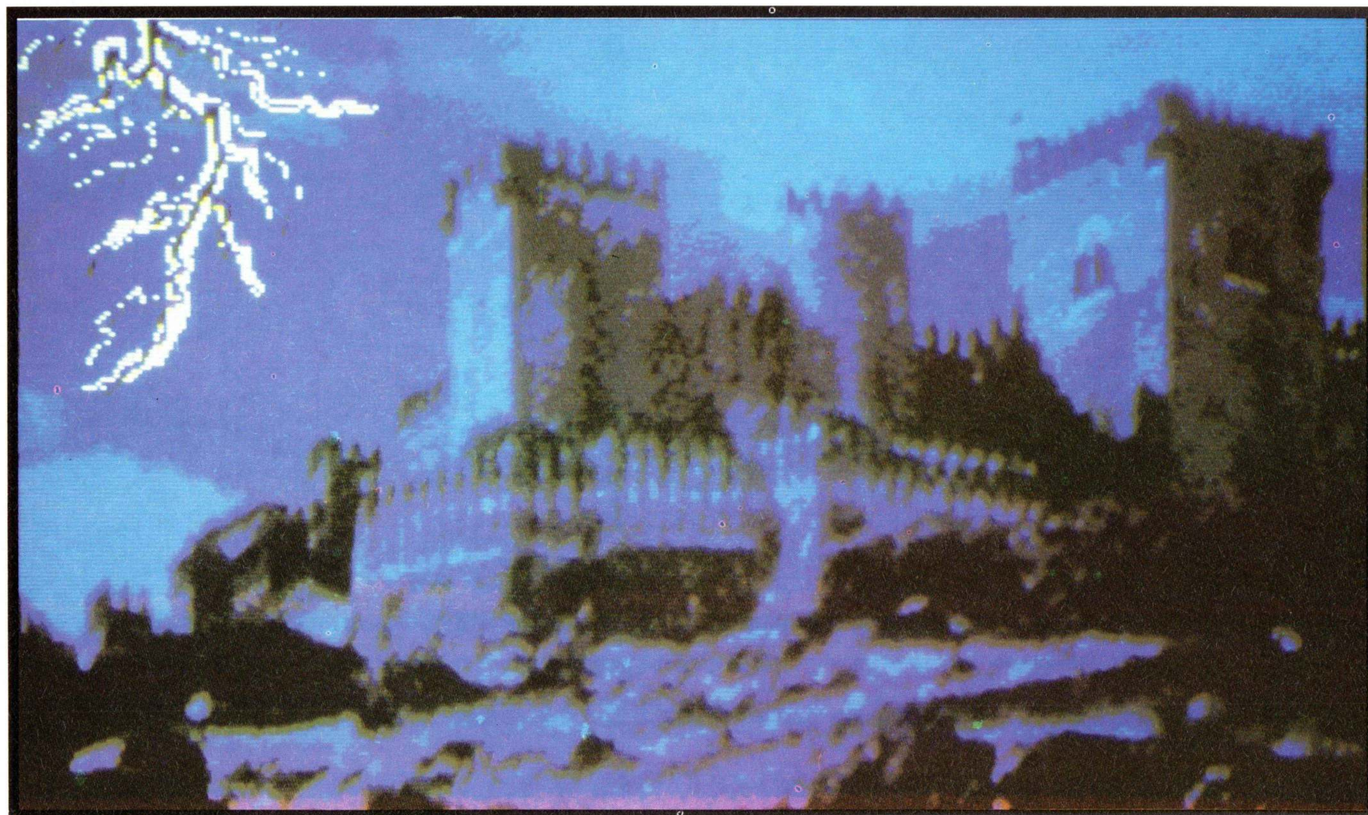
knapp 200 Seiten das Arbeiten mit Deluxe Paint II erklärt wird. Dieser Teil ist sicherlich nicht zu lang geraten, denn die Möglichkeiten, die dieses Malprogramm bietet, sind so vielfältig, daß man ihnen kaum gerecht werden kann.

Im vierten Teil legt der Autor Walter Friedhuber erst richtig los, denn die "Special Effects" sind angesagt. Einer der Hauptpunkte ist dabei die Schablonentechnik von DPaint, mit der sich eine Vielzahl von Effekten erzielen lassen. Weitere Themen sind der Hintergrund, perspektivisches Zeichnen und Schriftgestaltung. Dieses Kapitel wird von einer großen Zahl von Übungen begleitet, die die jeweiligen Effekte darstellen.

"Tips, Tricks und Utilities" zu Bildschirmfotografie, Einbinden von Fremdzeichensätzen und Digitalisieren von Fotovorlagen und andere interessante Themen werden aufgegriffen, bevor sich alles nur noch um Animationen dreht. Hier werden die Fähigkeiten von Deluxe Video ausgeschöpft. Vom Storyboard über Drehbuch, Entwurfsphasen und Background-Layout bis zum fertigen Video-Clip reicht die Palette. Wichtig sind dabei natürlich vor allem die Animationstechniken, die ausführlich in Bild und Wort erläutert werden. Am Ende hat man dann eine Menge Video-Clips und viele Ideen für eigene Projekte, die man auch gleich in die Tat umsetzen sollte.



Marilyn spricht: Animation durch Veränderung des Mundes.

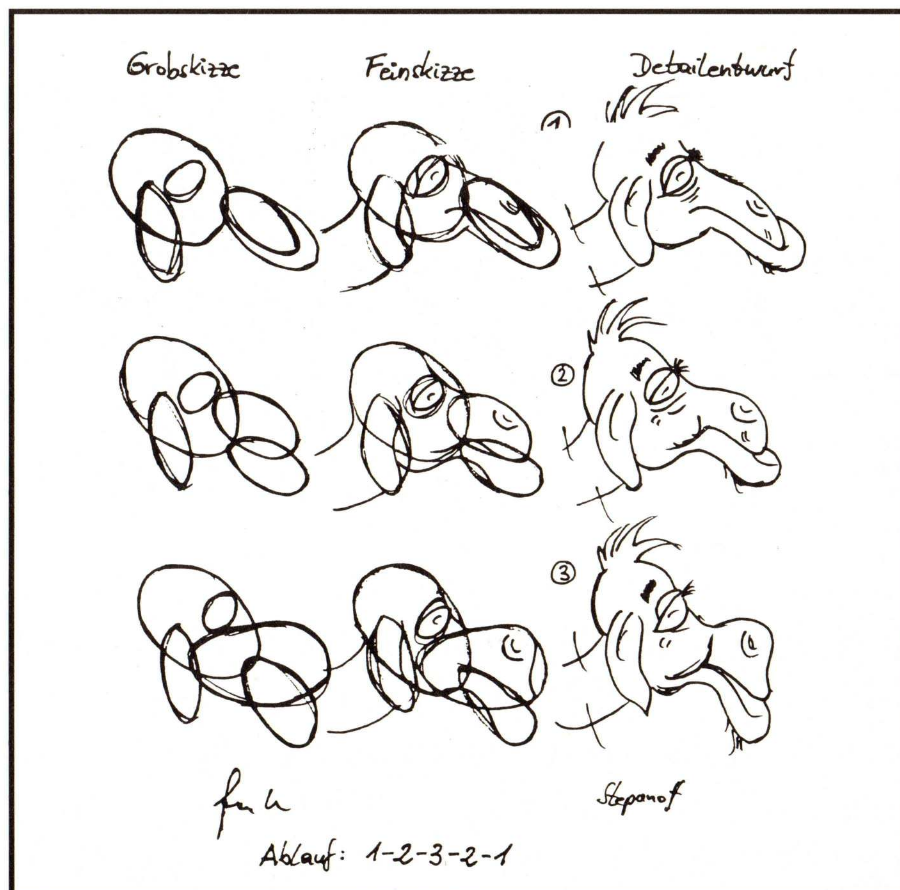


Stimmungsmanipulationen durch geschickte Farbwahl.

Überhaupt bietet dieses Buch eine unerschöpfliche Fundgrube mit vielen Anregungen für eigene Bilder und Projekte wie Video-Clips oder Comics. Die umfangreichen und ausführlichen Grundlagenteile sind für Grafikfans von hohem Wert, wenn ich auch dafür plädiert hätte, daß die Bilder ausschließlich von Hand gezeichnet werden sollten. Die Bilder des Buchs bzw. die schwarzweißen Hardcopys sind nämlich der große Schwachpunkt des Buches. Ihre Qualität ist zum Teil sehr schlecht und die Konvertierung nur unzureichend. Daraus ergibt sich fast die Notwendigkeit, die Begleitdisketten zu kaufen, die jedoch noch einmal je 25 DM kosten (Diskette 1 beinhaltet alle Übungen, die zweite die Animationen und Videos).

Trotzdem: Sehr empfehlenswert für Einsteiger und Fortgeschrittene, besonders wegen den umfangreichen Grundlagen und Anregungen.

Bezugsquelle:
Verlag Gabriele Lechner
Planegger Str. 1
8000 München 60
Preis: 58.- DM



Das Kamel in verschiedenen Entwürfen.



Von der Studie zum Bild.

VON CHRISTIAN MÖNCH

IFF-BILDER EINGEWICKELT

Transformer V 1.3

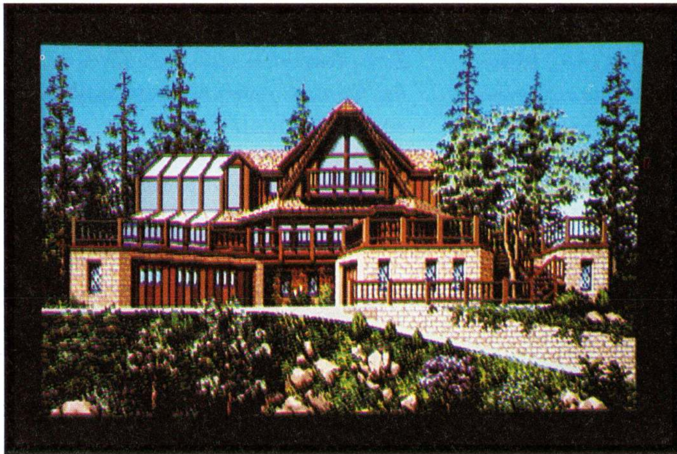
Bedienung des Programms: Das Programm wird vom CLI aus mit "Transformer <RETURN>" ohne Angabe von Parametern gestartet. Will man das Programm von der Workbench aus starten, muß sichergestellt sein, daß ein CLI-Window existiert. Andernfalls verfügt Transformer über keine Ein- und Ausgabemöglichkeiten und kann weder benutzt noch beendet werden.Nach dem Start wird nach dem zu bearbeitenden Bild gefragt, hier muß ein gültiger AmigaDos-Name eingegeben werden. Das Bild muß im IFF-Format vorliegen, das von fast allen Grafikprogrammen (z.b Deluxe-Paint oder Aegis-Images) erzeugt wird. Es können sämtliche Auflösungen inklusive des HAM-Modus verwendet werden, wobei sich dieser allerdings nicht empfiehlt, da es zu unerwünschten Streifen kommen kann. Kann das Bild nicht geladen werden, wird das Programm beendet. Ist das Bild erfolgreich geladen, wird zur Eingabe der für die Transformation notwendigen Daten aufgefordert.

Die ersten vier geforderten Werte beziehen sich auf den zu bearbeitenden Bildausschnitt. Hierbei gibt die X-Startposition den Ausgangspunkt auf der X-Achse und die X-Stopposition den Endpunkt auf der X-Achse an. Sollte die Stopposition kleiner als die Startposition sein, so wird das Bild von rechts nach links, also spiegelverkehrt, abgebildet. Das gleiche gilt für die Y-Start- und Stopposition.

Der nächste geforderte Wert gibt an, um welchen Winkel das Bild in X-Richtung um die Kugel "gezogen" werden soll. Ist der Winkel 360 Grad, wird das Bild vollständig in X-Richtung um die Kugel gezogen; dabei berühren sich der rechte und der linke Bildrand auf der Rückseite der Kugel. Oben Gesagtes gilt größtenteils auch für den Y-Winkel, dieser kann jedoch maximal nur 180 Grad betragen. Beträgt der X-Winkel 360 und der Y-Winkel 180 Grad, wird das Bild auf einer vollständigen Kugel abgebildet. Die Angabe sämtlicher Winkel erfolgt im Gradmaß.

Die folgenden drei Parameter *(xrot,yrot,zrot)* bezeichnen die Rotation der Kugel um x-, y- und z-Achse und müssen durch Komma getrennt eingegeben werden (z.B.: 30,40,50). Ist der Wert aller drei Parameter Null, blickt man von vorne auf eine aufrecht stehende, nicht gekippte Kugel. Auch hier erfolgt die Angabe der Winkel im Gradmaß.

Zu beachten ist, daß die Darstellung der Kugel ohne Perspektivtransformation erfolgt.



My home is...



... my world: Ein IFF-Bild um eine Kugel gewickelt.

Die Parameter *xoffset* und *yoffset* bestimmen den Bildschirmpunkt, an dem der Mittelpunkt der Kugel liegt (z.B.: *xoffset* = 0, *yoffset* = 0 ; die Kugel liegt in der oberen linken Ecke des Bildschirms).

Die nächsten beiden Parameter (*xradius* und *yradius*) bestimmen den Radius der abgebildeten Kugel, und damit ihre Größe. Dies geschieht folgendermassen:

1. Beträgt zum Beispiel der X-Winkel 180 Grad oder mehr, so gibt der X-Radius die Größe des Bildes in X-Richtung an.

2. Ist der X-Winkel kleiner 180 Grad (z.B. 2), so muß der X-Radius entsprechend groß gewählt werden, da er hier nicht mehr die Größe in Pixeln angibt, sondern mit dem Sinus des X-Winkels multipliziert wird. In diesem Fall kann man die X-Größe auf dem Bildschirm automatisch, gemäß des Y-Radius berechnen lassen. Dies geschieht durch Eingabe von Null bei der Frage nach dem X-Radius. Der berechnete X-Radius wird dann nach der Eingabe des Y-Radius angezeigt. Dasselbe gilt natürlich auch für den Y-Radius. Wird bei einer der beiden Größenangaben Null eingegeben, ist der Proportionalmodus gewählt, bei dem die X- und Y-Größe des Bildes auf dem Bildschirm gleich sind. Ist dieser Modus gewählt, und die Auflösung des Bildes beträgt 640x200 oder 320x400 Pixel (die Pixel sind dann doppelt so hoch wie breit, bzw. doppelt so breit wie hoch), wird die Eingabe einer Referenzgrösse erwartet. Hier kann entweder *x* oder *y* angegeben werden. Je nach Eingabe wird entweder der X-Radius oder der Y-Radius beibehalten, und der andere Radius so verändert, daß die Kugel genauso breit wie hoch ist. Am besten verdeutlicht man sich die Funktion der Referenzgröße an einem Beispiel: Angenommen, die Auflösung eines Bildes beträgt 320x400 Punkte (Lo-Res und Interlace), dann sind die einzelnen Pixel doppelt so breit wie hoch. Setzt man nun den X-Proportionalmodus (durch Eingabe von Null als X-Radius), und den Y-Radius gleich 100, berechnet das Programm den entsprechenden X-Radius (in diesem Fall ungefähr 98), und fragt nach der Referenzgröße. Gibt man hier "y" ein, bewirkt dies, daß der Y-Radius beibehalten wird, der X-Radius jedoch durch Zwei dividiert wird. Man erhält also ein kleineres Bild. Hätte man "x" als Referenzgröße angegeben, so wäre der X-Radius beibehalten und der Y-Radius mit Zwei multipliziert worden, das Bild also größer geworden.

Sind X- und Y-Winkel kleiner 180 Grad, so müssen X- und Y-Radius im Zweifelsfall durch Probieren ermittelt werden. Punkte, die im berechneten Bild außerhalb des Bildschirmfensters liegen, werden nicht gesetzt.

Der letzte Parameter (Modus) bestimmt, ob die Kugel durchsichtig (0) oder undurchsichtig (1) ist. Durchsichtig bedeutet, daß man Punkte, die auf der Rückseite der Kugel liegen, sehen kann, wenn sie nicht durch Punkte auf der Vorderseite verdeckt werden (man kann also die Innenseite der Kugel sehen). Eine undurchsichtige Kugel verdeckt alle auf ihrer Rückseite liegenden Punkte.

Sind alle Parameter eingegeben, wird die Berechnung des Bildes gestartet.

Transformer öffnet beim Start zwei eigene Screens. Der erste wird zum Laden des Ausgangsbildes, der zweite zur Darstellung des transformierten Bildes benötigt.

Die Berechnung der Kugel kann unterbrochen werden, indem man den zweiten Screen einfach mit der Maus nach unten zieht. Das Programm fragt dann, ob die Transformation beendet werden soll. Gibt man hier "j" ein, kehrt man zum Hauptmenü zurück; bei "n" wird die Berechnung fortgesetzt.

Ist die Berechnung eines Bildes beendet, springt das Programm in das Hauptmenü. Hier bestehen folgende Optionen:

"e" - Beenden des Programms.

Der benutzte Speicherplatz wird freigegeben, und das Programm wird beendet.

"s" - Speichern des Bildes im IFF-Format.

Das Programm fragt hier nach dem Namen, unter dem das Bild gespeichert werden soll. Es sind alle gültigen AmigaDos-Namen erlaubt. Wird kein Dateipfad angegeben, wird das Bild im aktuellen Verzeichnis gespeichert.

"n" - Neustart der Berechnung mit neuen Parametern.

Es wird eine neue Berechnung mit dem vorhandenen Ausgangsbild gestartet, wobei die Parameter neu eingegeben werden.

"l" - Laden eines Bildes.

Es wird ein neues Ausgangsbild geladen.

Allgemeine Hinweise zur Arbeit mit Transformer:

Bei der Eingabe der Parameter ist darauf zu achten, daß die geforderte Eingabemaske (z.B. Komma zwischen *xrot*, *yrot* und *zrot*) eingehalten wird, da die Aztec-I/O- Routinen sonst fehlerhaft arbeiten und keine weitere Eingabe mehr zulassen. In dem Fall muß das Programm mit Control-"c" abgebrochen werden, wobei jedoch die beiden von Transformer geöffneten Screens nicht wieder geschlossen werden, da die transformereigene "Endroutine" nicht angesprungen wird.

Sollten während der Arbeit mit Transformer Fehler auftreten, werden diese gemeldet und das Programm beendet, wobei natürlich alle geöffneten Screens, Librarys und Files geschlossen werden.

Die Rechenzeit, die zur Transformation benötigt wird, hängt nicht von der Größe des Ausgangsbildes ab, vielmehr richtet sie sich nach der Größe des transformierten Bildes. Will man also seine eingegebenen Parameter überprüfen, macht man das am besten mit einer kleinen Größe, da die Berechnung dann relativ schnell beendet ist.

Interessante Ergebnisse erzielt man, wenn z.B. der X-Winkel 240 Grad und der Y-Winkel 1 Grad beträgt. Setzt man die X-Größe auf 40 und läßt die Y-Größe im Proportionalmodus berechnen, wird das Bild auf einer Röhre dargestellt, die ungefähr 40x40 Pixel groß ist.

Der Eindruck der Röhre entsteht dadurch, daß eine Krümmung des Bildes um 1 Grad in Y-Richtung auf dem Bildschirm nicht zu erkennen ist.

Ist einer der Winkel (oder beide) kleiner als 90 Grad und der entsprechende Radius groß, kann es bei der Rotation passieren, daß das Bild außerhalb des Bildschirmfensters liegt, da sich das Bild immer auf der Oberfläche einer Kugel befindet, die in diesem Fall sehr groß ist. Um hier Abhilfe zu schaffen, ist es notwendig, X- und Y-Offset entsprechend anzugleichen. Hierbei geht man am besten so vor, daß man das Bild mit wachsender Größe berechnen läßt; so kann man die Lage des Bildes am besten verfolgen und die beiden Offsets angleichen.

Sollte bei einer Operation (Laden, Speichern u.s.w.) ein Fehler auftreten, so wird das Programm verlassen. Etwaige Bilder gehen dabei verloren.

Vielleicht wundert sich mancher über die beim Amiga unübliche Eingabe der Parameter, die vollkommen ohne Pull-Down-Menüs und Gadgets geschieht. Auf diesen Komfort wurde jedoch bewußt verzichtet, da Menü- und Gadgetdefinitionen den Quelltext um mindestens 30% vergrößert hätten.



Das Ursprungsbild...



...kann auch um eine Röhre gewickelt werden.

Hinweise zu den IFF-Routinen:

Die Lade- und Saveroutinen des Transformers bearbeiten IFF-Bilder, die als vollständiges Bild gespeichert wurden, also keine "Brushes", wie sie Deluxe-Paint speichert. Brushes kann man jedoch in gewissem Umfang durch die Angabe der Start- und Stoppositionen selbst erstellen.

Es werden sowohl komprimierte als auch unkomprimierte Bilder geladen. Gespeichert werden die Bilder allerdings immer komprimiert, was besonders bei kleinen Darstellungen erheblich Diskettenplatz spart.

Folgende IFF-Chunks und Daten bleiben unberücksichtigt:

1. Colorcycle-Chunks (z.B. "CRNG" bei Deluxe-Paint).

Möchte man Cycle-Colors benutzen, müssen die Bilder mit einem Malprogramm nachbearbeitet werden.

2. Masking (enthalten im BMHD-Chunk).

Masken werden beim Laden übergangen, und nicht abgespeichert.

3. X- und Y-Offset (enthalten im BMHD-Chunck).

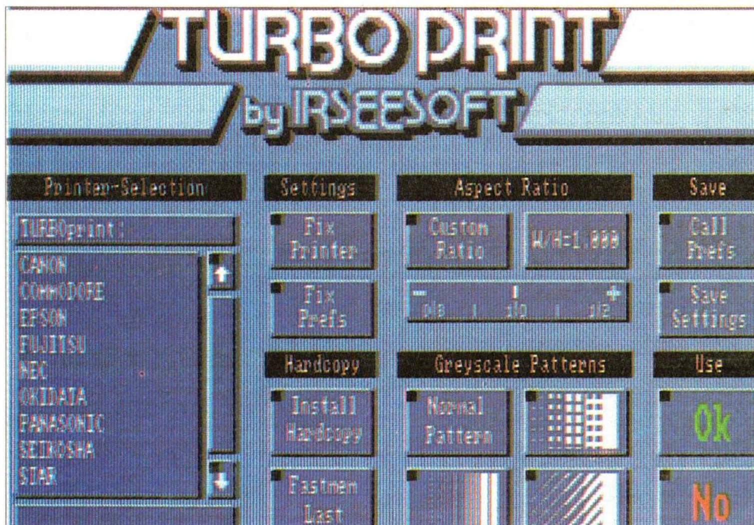
4. *transparentcolor* (BMHD-Chunck)

5. *xaspect* und *yaspect* (BMHD-Chunk).

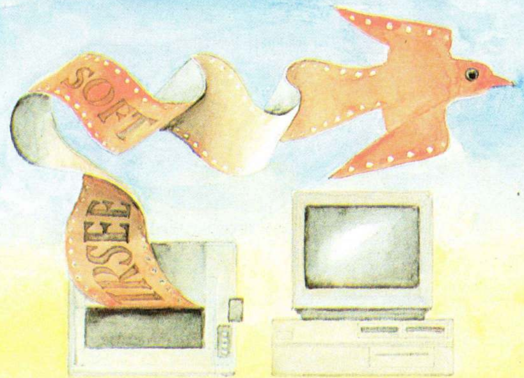
Unberücksichtigte Daten werden zwar gelesen, jedoch nicht auf Diskette geschrieben; so hat ein Bild z.B. keine "CRNG"-Chunks. Die unberücksichtigten Daten stellen jedoch im allgemeinen keine Beeinträchtigung der Arbeit mit dem Transformer dar.

Hinweise zum Abtippen des Programms:

Das Programm Transformer ist mit dem Aztec C-Compiler 3.4a compiliert worden. Der Quelltext von Transformer kann mit jedem ASCII-Texteditor (z.B. ed , micro-emacs) erstellt werden. Hat man den Quelltext



TURBO PRINT



Das revolutionäre ***AMIGA*** Druckprogramm
von IrseeSoft
schnell, kompatibel und vielseitig - für perfekte Bilder ·
resetfest mit Hardcopy- und Nofastmemfunktion

TURBOprint –
Das komfortable Druckerspeeder-Paket für Ihren AMIGA

- bis zu 10x schnellerer Grafikdruck durch 100% 68000-Assembler-Programmierung
- unterstützt alle Bildschirmauflösungen des Amiga
- bessere kontrastreiche Bilder durch vier wählbare Farbumsetzungen
- druckt 4096 Farben im HAM-Modus jetzt auch ausschnittweise und gedreht
- spezielle Turbotreiber für alle Druckdichten Ihres Matrix-, Tintenstrahl-, Thermo- oder Laserdruckers
- resetfeste Hardcopyfunktion druckt jeden Bildschirm aus
- keine Probleme mehr mit Speichererweiterungen durch verbessertes, resetfestes Nofastmem
- ohne Umstellung wie gewohnt weiterarbeiten
- vollkompatibel zur Amiga-Software
- wird resetfest im Speicher installiert
- arbeitet unbemerkt im Hintergrund
- kein Umkopieren auf Ihre Software notwendig
- ausführliches deutsches Handbuch
- läuft auf Amiga 500, 1000 und 2000

DM 89,-

Vertrieb:

IrseeSoft SPCS Heinz Donhauser
Grüntenstraße 6
8951 Irsee
Telefon (0 83 41) 7 43 27

PDC GmbH
Louisenstraße 115
6300 Bad Homburg
Telefon (0 61 72) 2 47 48 / 2 07 99

AMIGA Hard- und Software Aktion

Butcher 2.0	DM 69,—
Pixmate	DM 139,—
Digi Paint	DM 129,—
Videoscap 3 D	DM 299,—
Aegis Draw Plus	DM 409,—
Aztec C Developer V 3.6	DM 449,—
Amiga Entwickler-Handbücher (4 Stück), Libr. & Dev., Intuition, Hardware, Exec	DM 249,—
Amiga kommentiertes ROM-Listing 1 + 2 je Bd.	DM 69,—

Fordern Sie unseren ausführlichen Hard- und Softwarekatalog an.



Ralf Tröps, Computertechnik
Pingsdorfer Straße 141, 5040 Brühl
Telefon (0 22 32) 1 30 63 + 4 71 05

---

2 Jahre Garantie auf Hardware und geprüfte Software mit Zertifikat, beides in bester Qualität unterscheiden uns zu Billigangeboten

AMIGA aktuell — Dieter Hieske
Ladenlokal Schillerstraße 36
6788 Ludwigshafen-Oggersheim
Telefon (06 21) 67 31 05

Öffnungszeiten: Montag—Freitag 9.30—12.00 Uhr, Montag—Freitag 12.00—18.00 Uhr, Samstag 9.00—13.00 Uhr. Danach Anrufbeantworter.

PUBLIC DOMAIN SOFTWARE KOPIERSERVICE

lieferbar sind: Fish 1-135/
Panoram 1-51/Faugl-53/Amicus 1-20
Amusel-3/Auge 4000 1-14/Taifun 1-40
Tornado 1-30/RW Disk 1-30/ES Soft 1-55
Kick 1- / TBAG 1-7/Chiron 1-40 u.viele mehr

Kopierpreise per Stueck wenn wir die Disketten 2DD mitliefern
1-9 7,00/10-19 6,50/20-29 6,00/30-49 5,50
50-99 5,00/100-199 4,50/ ab 200 4,00

Kopierpreise wenn Sie uns Ihre Disketten per Einschreiben zusenden

1-9 4,00/10-19 3,50/20-29/3,00/30-49 2,50
50-99 2,00/100-199 1,50/ ab 200 1,00

Kopierpreise decken nur die Kosten fuer Laufwerke und Anschaffungskosten

INHALTSVERZEICHNIS Public Domain kostenlos

Software Anwender

Sculpt 3D 182,90 / Animate 3D 348,00
Silver 279,00 / Aegis Draw 175,95
Aegis Imp. 164,90 / Deluxe P. II 249,00
Deluxe Vid. 249,00/ Forms in Flight 158,95
Digi View PAL 2.0 640x512 DM 333,00
X-CAD 998,00 / Videoscape3D 328,00
TV-Show 198,00 / TV Text 229,00
Pagefliper 169,00 / Calligrapher 201,50
Aegis Video Titler 198,00
AC Basic Compiler 377,00 / Lattice C 4.0 403,90
Profimat 99,00 / Butcher 2.0 65,90
Marauder II 72,90 / Metacomco Shell 113,95
Aegis Sonix 132,50 / Dynamic Drums 132,50
Instant Music 98,00 / Perfekt Sound 160,00
City Desk 269,00 / Vizawrite 1.05 198,00
Publisher 1000 353,90 / Prowrite 221,50
Suberbase 249,00 / Logistix 384,50

Software Spiele

Barbarian 67,50 / Feud 25,90
Rallye Master 25,90 / Hellowoon 63,00
Dark Castle 67,50 / Crazy Cars 44,00
Crystal Hammer 35,95 / Winter Olymp 88 54,50
Testdrive 76,50 /Ninja Mission 27,50
Clever&Smart ? / Harrier Command ?
Jagd. r. Octob. 67,50 / Jinxter 72,00
Articfox 59,80 / Fire Power 68,95
Jinks 51,90/ Garrison II 59,90
Int. eagles nest 56,50/ OGRE 91,00
Emerald Mine 25,95 / Pink Panther ?
?=lieferbar,aber bei Druck noch kein Preis
anrufen .STAENDIG NEUHEITEN lieferbar

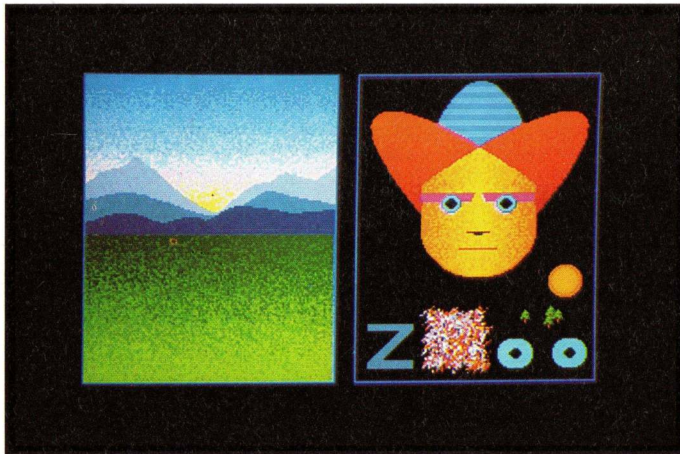
PREISLISTE kostenlos

Hardware A500: 512Kb abschalt. m. Uhr akkugep. DM 229,00
A500: 2MB extern DM 888,00
A1000: 2MB extern DM 888,00
A2000: 2MB extern +SCSI DM 966,00

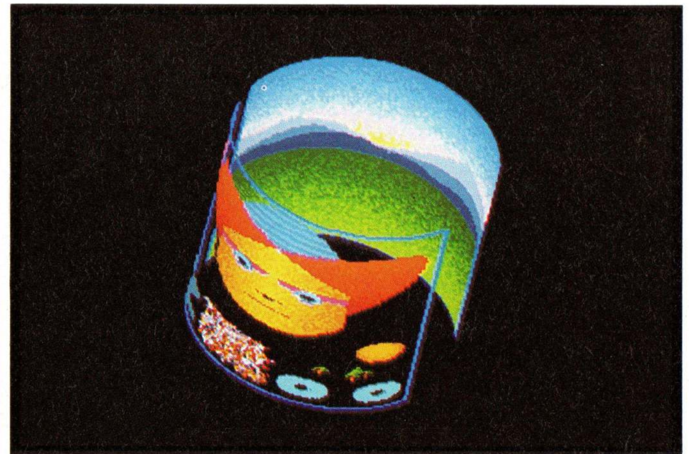
Laufwerk ext. 3,5" DM 329,00
Laufwerk ext. 5,25" DM 339,00
Laufwerk A2000 int. DM 339,00
Bootselektor DF1 DM 19,90
Adapter f. 3Laufw. a 1 Port DM 59,00
Festplatte 20MB A2000 DM 1398,00
A1000 DM 1888,00 A500 DM 1898,00

Versand per NN + DM 8,00
Versand per UPS
Ab DM 500.- keine Versandkosten

PREISLISTE und PUBLIC DOMAIN INHALTSVERZEICHNIS kostenlos anfordern



Ungesetzte Farben (schwarz) im Ursprungsbild...



... können in der Transformation durchscheinend dargestellt werden.

eingegeben und unter dem Namen "Transformer.c" gespeichert, wird er mit folgendem Befehl compiliert: "cc transformer.c -E200". Die Option "-E200" veranlaßt den Compiler, mehr Speicher für Ausdrücke zu reservieren. Sie verhindert, daß ein "Out of expression space"-Fehler auftritt. Tritt dieser Fehler bei Ihrer Compilerversion denoch auf, kann er durch Vergrößern der Zahl "200" hinter "-E" behoben werden.

Nachdem Compiler- und Assemblerdurchlauf erfolgreich beendet sind, kann das Programm mit "ln -o Transformer Transformer.o -ls -lm -lc" gelinkt werden. Es ist darauf zu achten, daß m.lib (-lm) vor c.lib (-lc) angegeben wird, da das Programm in einigen Prozeduren "float"-Werte als Parameter übergibt. Ist der Linkerdurchlauf beendet, steht das Programm als ausführbares File unter dem Namen "Transformer" auf Diskette.

Berechnung des Bildes "sachscastle.trans"

Format:
Lo-Res, Non-Interlace (320x200)
Parameter:
xstartposition = 0
xstopposition = 320
ystartposition = 0
ystopposition = 200
xwinkel = 230
ywinkel = 170
xrot,yrot,zrot = 170,20,200
xoffset,yoffset = 160,100
xradius = 0 (X-Proportionalmodus)
yradius = 100
modus = 0

Berechnung des Bildes "stencilset.trans"

Format:
Lo-Res, Noninterlace (320x200)
Parameter:
xstartposition = 0
xstopposition = 320
ystartposition = 0
ystopposition = 200
xwinkel = 360
ywinkel = 1
xrot,yrot,zrot = 30,10,260
xoffset,yoffset = 160,100
xradius = 100
yradius = 0 (Y-Proportionalmodus)
modus = 0

```
/*************************************************************************
 *
 *                      IFF-Transformer V1.3
 *                      --------------------
 *
 *
 *      Laedt ein IFF-Bild, stellt das Bild auf einer Kugeloberfaeche
 *      dar, und speichert das transformierte Bild als IFF-File
 *
 *      (C) 11/87 Christian Moench
 *
 *                Kellerbornstr.15
 *             6000 Frankfurt a. Main
 *
 *************************************************************************/

#include <exec/types.h>
#include <stdio.h>
#include <functions.h>
#include <graphics/gfx.h>
#include <graphics/gfxbase.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <math.h>

/* Macrodefinitionen zu den Rechenroutinen */

#define ABS(x) ((x) < 0.0) ? (-(x)) : (x)
#define PI 3.141592654
#define TOBOGEN(a) ((a)*PI/180.0)          /* wandelt Grad- in Bogenmass */
#define TOGRAD(a) ((a)*180.0/PI) /* wandelt Bogen- in Gradmass */
/* maximal zulaessiger Abtastwinkel fuer
   eine gegebene Kugelgroesse */

#define MAXWIN(s) ((asin(.6/(s))<0) ? (PI+asin(0.6/(s))) : (asin(.6/(s))))

/* groesster Sinuswert im Bereich von 0-winkel */
#define BIGESTWIN(s) (double)((s >= 90.0) ? (1.0) : (sin(TOBOGEN(s))))

#define VOLL 1              /* Kugel undurchsichtig */
#define LEER 0              /* kugel durchsichtig   */
#define MONITOR 1.5         /* siehe getprop */
#define ASPECT aspects[(meinbmhd.w/meinbmhd.h)]

/* Definitionen zu IFF-Files
Macro  zum Errechnen der Id aus vier Buchstaben */
```

```
#define IDVAL(a,b,c,d) ((LONG)a << 24L | (LONG)b << 16L |  c << 8 | d)

/* Id's der einzelnen Chuncks */

#define ID_FORM  IDVAL('F','O','R','M')
#define ID_ILBM  IDVAL('I','L','B','M') /* Interleave Bitmap */
#define ID_BMHD  IDVAL('B','M','H','D') /* Bitmap Header */
#define ID_CMAP  IDVAL('C','M','A','P') /* Colormap */
#define ID_BODY  IDVAL('B','O','D','Y') /* Bilddaten */

/* BitMapHeader-Structure aufgebaut wie der "BMHD"-Chuncks. Dieser
Chunck enthaelt alle wichtigen Informationen ueber das zu ladende Bild  */

typedef struct {
 UWORD w,h;                /* Breite,Hoehe */
 WORD x,y;                 /* x- und yoffset beim Laden */
 UBYTE nPlanes;            /* Anzahl der Bitmaps */
 UBYTE masking;            /* Maske ? (nicht beruecksichtigt) */
 UBYTE compression;        /* gepackt ? */
 UBYTE pad1;               /* die folgenden Werte */
 UWORD transcol;           /* werden nicht beruecksichtigt, */
 UBYTE xasp,yasp;          /* sind aber im BMHD-Chunck und  */
 WORD paw,pah;             /* werden gelesen                */
 } BitMapHeader;


BitMapHeader meinbmhd;   /* Speicher fuer "BMHD"-Struktur belegen */

UBYTE farben[96];          /* Array zum Einlesen der Farben */
ULONG colors;              /* Anzahl der Farben des jeweiligen Bildes */
UBYTE *loc_bmhd; /* Pointer auf "BMHD"-Struktur */
UBYTE *loc_cmap; /* Pointer auf Farbarry */

struct GfxBase *GfxBase;          /* System pointer Grafik */
struct IntuitionBase *IntuitionBase;     /* System pointer Intuition (wird zum
                                        Oeffnen der Screens benoetigt) */

/* Die folgenden Pointer sind jeweils fuer das Ausgangsbild und das trans-
formierte Bild definiert */

struct RastPort *rp,*rp2;
struct ViewPort *vp,*vp2;
struct Screen *s=NULL;
struct Screen *s2=NULL;

/* Newscreen Struktur wird vor dem Oeffnen der Screens vom Program an das zu
ladende Bild angepasst (Werte aus dem "BMHD"-Chunck) */

struct NewScreen ns = {0,0,320,200,5,
                   0,1,
                   0,
                   CUSTOMSCREEN,
                   NULL,
                   NULL,
                   NULL,
                   NULL };

/* Speicherreservierung und Initialisierung der Rotationsmatrix zum Drehen der
Kugel */

double m[3][3] = {{1.0,1.0,1.0},
           {1.0,1.0,1.0},
           {1.0,1.0,1.0}};

double aspects[4] = {2.0 , 1.0 , 0.5 , 0.5 };

char zeichenpuffer[80]; /* Puffer fuer die Funktion "scanf" */
FILE *cleanup;              /* wird von "cleanexit()" benoetigt, um offene Files
                       zu schliesen */


/*-------------------------------------------------------------------------*/
/*                              Disk - Routinen                            */
/*-------------------------------------------------------------------------*/

/*-------------- hole Byte von spezifizierten File ------------------------*/

BYTE getbyte(fp)

 FILE *fp; /* Pointer auf File Struktur */

{
 BYTE erg;

    erg = getc(fp);       /* Lese Byte */
    if(feof(fp)) {        /* End of File ? */
 cleanexit("Unerwartetes Fileende !");
    }
    if(ferror(fp)) {     /* Lesefehler ? */
 cleanexit("Fehler beim Lesen aufgetreten !");
    }
    return(erg); /* kein Fehler: gebe Byte zurueck */
}


/*------------ hole Langwort aus spezifizierten File ----------------------*/
```

```

ULONG getlong(fp)

 FILE *fp;        /* Pointer auf File-Struktur */

{

 ULONG len;

 len = 0L;
 len = (ULONG)IDVAL(getbyte(fp),getbyte(fp),getbyte(fp),getbyte(fp));
 return(len);

}


/*--------------- schreibe Byte in spezifiziertes File --------------------*/

void putbyte(fp,b)

 FILE *fp;        /* Pointer auf File-Struktur */
 BYTE b;          /* zu schreibendes Byte      */

{

    putc(b,fp);
    if(ferror(fp)) {      /* Fehler beim Schreiben ? */
 cleanexit("Fehler beim Schreiben aufgetreten !");
    }
}


/*------------- schreibe Langwort in spezifiziertes File ------------------*/

void putlong(fp,l)

 FILE *fp;        /* Pointer auf File-Struktur */
 LONG l;          /* zu schreibendes Langwort  */

{

    putbyte(fp,(UBYTE)((l >> 24L) & 0xff));              /* Byte Nr. 0 */
    putbyte(fp,(UBYTE)((l >> 16L) & 0xff));              /* Byte Nr. 1 */
    putbyte(fp,(UBYTE)((l >>   8) & 0xff));              /* Byte Nr. 2 */
    putbyte(fp,(UBYTE)( l         & 0xff));              /* Byte Nr. 3 */

}


/*------------------------ Lade IFF-Bild ----------------------------------*/
loadpic(name)

 char *name;     /* Pointer auf Filenamen */

{

    FILE *fp;
    LONG id;
    void lookforpic();

    if((fp = fopen(name,"r")) == 0) {            /* File nicht geoffnet ? */
 cleanup = NULL;                           /* kein File offen */
 cleanexit("File nicht gefunden !");      /* beenden        */
    }
    cleanup = fp;                   /* File "fp" ist offen */
    id = getlong(fp);               /* hole Id */
    if(id != ID_FORM) {             /* ist IFF-File ? */
 cleanexit("Kein IFF-File !");
    }
    lookforpic(fp);                 /* hole Bild */
    fclose(fp);                     /* schliese File */
    cleanup = NULL;                 /* kein File offen */
}


/*---------------- Ueberlese unbekannten Chunck ---------------------------*/

overreadchunk(fp,l)

 FILE *fp;        /* Pointer auf File-Struktur */
 ULONG l;      '  /* Chuncklaenge */

{

 ULONG i;
 BYTE dummy;

    for(i=0;i<l;i++) {            /* ueber gesamte Laenge */
 dummy = getbyte(fp);    /* Byte wird nicht benoetigt */
    }
    if(l & 0x1L) {                /* Laenge ungerade ? */
 dummy = getbyte(fp);    /* ja,lese pad-Byte */
    }

}

```

```c
/*------- prueft ob ein IFF-File vorliegt und liest ein vorhandenes Bild ----*/

void lookforpic(fp)

 FILE *fp;        /* Pointer auf File-Struktur    */

{

 BOOL body;
 LONG length,id,i;
 BYTE dumm;

    body = FALSE;                   /* Bilddaten noch nicht erreicht */
    length = getlong(fp);           /* hole Chuncklaenge */
    id = getlong(fp);               /* hole Chunckid */

    if(id != ID_ILBM) {             /* Bilddatei enthalten ? */
 cleanexit("Kein Bild gefunden !",fp);
    }

    while(!(body)) {                /* Solange keine Bilddaten gelesen wurden */
 id     = getlong(fp);   /* hole id */
 length = getlong(fp);   /* hole Laenge */

 switch(id) {            /* bearbeite Chunks */

     case(ID_BMHD) : {   /* BitMapHeader ? */

         loc_bmhd = (UBYTE *)(&meinbmhd);
         for(i=0;i<length;i++) {
             *loc_bmhd++ = getbyte(fp);
         }
         if(length&1)
             dumm = getbyte(fp);
         computebmhd();
         break;
         }

         case(ID_CMAP) : {       /* colormap ? */
             loc_cmap = farben;
             for(i=0;i<length;i++) {
                 *loc_cmap++ = getbyte(fp);
             }
             colors=length;
             if(length&1) {
                 dumm = getbyte(fp);
             }
             computecmap();
             break;
         }

         case(ID_BODY) : {       /* body ? */
             readbody(fp);
             body = 1;           /* Bild ist gelesen */
             break;
         }
         default       : {       /* unbekannter Chunck */
             overreadchunk(fp,length);
             break;
         }

 }

    }

}


/*----------- Oeffnet Screen mit den gelesene Parametern -----------------*/

computebmhd()
{
 LONG i;


    /* Anpassen der Newscreen-Struktur an das glesene Bild */

    ns.ViewModes = 0;
    ns.Depth = meinbmhd.nPlanes;
    if(ns.Depth == 6) {
 ns.ViewModes |= HAM;
    }

    if(meinbmhd.w > 400) {
 ns.ViewModes |= HIRES;
    }
    ns.Width = meinbmhd.w;

    if(meinbmhd.h > 256) {
 ns.ViewModes |= LACE;
    }
    ns.Height = meinbmhd.h;

    /* Oeffne Screen 1, zum Laden des Bildes */

    if(s) {
 CloseScreen(s);
    }

    s = (struct Screen *) OpenScreen(&ns);
    if(s == NULL) {
 cleanexit("Screen no.1 kann nicht geoeffnet werden !");
    }

    /* Oeffne Screen 2, zum Transformieren des Bildes */

    if(s2) {
 CloseScreen(s2);
    }

    s2 = (struct Screen *) OpenScreen(&ns);
    if(s2 == NULL) {
 cleanexit("Screen no.2 kann nicht geoeffnet werden !");
    }

    /* Hole Viewports der beiden Screens, zum Setzen der Farben */
    /* Hole Rastports der beiden Screens, fuer Grafikfunktionen  */

    vp = &s -> ViewPort;
    rp = &s -> RastPort;

    vp2 = &s2 -> ViewPort;
    rp2 = &s2 -> RastPort;

    ScreenToFront(s);    /* Screen auf dem geladen wird kommt nach vorn */
    /* kann bei LATTICE "Screentofront" (mit kleinem t) heisseen */

}


/*--------------- setze Farben der gelesenen Colormap ---------------------*/

computecmap()

{
 LONG i,c,red,green,blue;

    loc_cmap = farben;            /* Pointer auf Array "farben" */
    c=colors/3L;          /* eine Farbe besteht aus drei Bytes */

    for(i=0;i<c;i++) {            /* ueber alle Farben */

 red = (LONG)(*loc_cmap++ >> 4);
 green = (LONG)(*loc_cmap++ >> 4);
 blue = (LONG)(*loc_cmap++ >> 4);

 SetRGB4(vp,i,red,green,blue);   /* setze Farben von Screen 1 */
 SetRGB4(vp2,i,red,green,blue);  /* setze Farben von Screen 2 */

    }

}


/*--------- dekommpriniert eine Bildzeile (Dpaint II und AegisImages) --------*/

UBYTE *unpackrow(bt,l,fp)

 UBYTE *bt;       /* Puffer fuer dekompriemierte Zeile, in diesem Fall
                     die Bitmap von Screen Nr. 1 */
 UWORD l;         /* Laenge der Zeile in Bytes */
 FILE *fp;        /* Pointer auf File-Struktur */

{

 BYTE n;
 UWORD i,j,k,count;

    count = 0;
    if(meinbmhd.compression == 1) {       /* ist komprimiert ? */
        while(count < l) {                /* ueber alle Bytes */
     n = getbyte(fp);
     if(n<0) {                    /* wiederholte Bytes ? */
         if(n > -128) {           /* kein Nooperation (-128) ? */
             k = getbyte(fp);    /* hole zu wiederholendes Byte */
             for(j=0;j<((-n)+1);j++) {
                 *bt++ = k;      /* wiederhole Byte -n+1 mal */
                 count ++;       /* schreibe Byte in Puffer
                                    und erhoehe Zaehler */
             }
         }
     }
     else {                       /* naechsten n+1 Bytes Lesen */
         for(j=0;j<(n+1);j++) {  /* ueber n+1 */
             *bt++ = getbyte(fp);        /* hole Byte */
             count++;                    /* schreibe es  in Puffer und
                                            erhoehe Zaehler */
         }
     }

 }
    }
    else {                                       /* nicht kompriemiert */
```

```
   for(j=0;j<l;j++) {                /* lese l Bytes */
      *bt++ = getbyte(fp);           /* und schreibe sie in Puffer */
   }
     }
     return(bt);            /* gebe neue Position im Puffer zurueck */
 }
 /*------ holt Anzahl der gleichen Bytes eines Arrays (hier: Bitmap)---------*/
 BYTE getsame(src,max)
  BYTE *src;       /* Array Position */
  WORD max;        /* maximale Anzahl der zu lesenden Bytes */
 {
  BYTE l,c,f,*hsrc;
     c=0; .
     f=0;
     hsrc = src+1;
     for(l=0;l<max;l++) { /* ueber die maximale Anzahl */
  /* vergleiche Byte mit Nachfolger */
  if((*src == *hsrc) && (f==0)) { /* ist gleich */
      c++;                           /* erhoehe Zaehler */
      src++;                         /* naechsten zwei Bytes */
      hsrc++;
  }
  else {  /* ist ungleich */
      f=1;          /* Flag setzen, so das keine weiteren Paare ge-
                       zaehlt werden */
  }
     }
     if(c==0) {            /* kein Paar ? */
  c=-1;
     }
     return(c);
 }
 /*------------ hole Anzahl der verschieden Bytes eines Arrays ------------*/
 BYTE getdifferent(src,max)
  BYTE *src;       /* Arbeitsweise analog zu getsame() */
  WORD max;
 {
  BYTE l,c,f,*hsrc;
     c=0;
     f=0;
     hsrc = src+1;
     for(l=0;l<max;l++) {
  if((*src!=*hsrc) && (f==0)) {
      c++;
      src++;
      hsrc++;
  }
  else {
      f=1;
  }
     }
     if(c==0) {
  c=-1;
     }
     return(c);
 }
 /*- komprimiert x Zeichen eines Arrays src, legt diese in das Array dest --*/
 LONG packrow(src,dest,x)
  BYTE *src,*dest;         /* source- und destinationArrays */
  BYTE x;                  /* maximale Bytezahl */
 {
  BYTE c,dif,sam,i;
  LONG l;
     l=0;
     c=0;
     while(c<x) {          /* ueber alle Bytes */
  sam=getsame(src,x-c);            /* hole Anzahl gleicher und */
  dif=getdifferent(src,x-c);       /* ungleicher Bytes */
  if(sam!=-1) {   /* gleiche Bytes */
      l+=2L;       /* Laenge des Ziel-Arrays wird um zwei groesser */
      c+=sam;      /* Anzahl der gelesenen Bytes wird um die Anzahl
                      der gleichen Bytes groesser */
      *dest++ = -(sam-1);  /* schreibe -Anzahl-1 in Ziel-Array */
      *dest++ = *src;       /* schreibe zu wiederholendes Byte */
      src+=sam;   /* Position im Source Array wird um die Anzahl
                     der gleichen Bytes groesser */
  }
  if(dif!=-1) {
      *dest++ = dif-1;    /* schreibe Anzahl-1 in Ziel-Array */
      l++;                /* Laenge des Ziel-Arrays wird
                             um eins groesser */
      for(i=0;i<dif;i++) {        /* ueber ungleiche Bytes */
          c++;                    /* gelesene Bytes wird
                                     um "dif" groesser */
          l++;                    /* Laenge des Zielarrays wird
                                     um "dif" groesser */
          *dest++ = *src++;       /* schreibe die ungleichen Bytes ins
                                     Ziel-Array */
      }
  }
     }
     return(l);
 }
 /*-------------------- holt Bitmap-Adrressen ----------------------------*/
 UBYTE *getbitmap(num)
  LONG num;        /* welche Bitmap */
 {
  UBYTE *plane;
  plane = s->BitMap.Planes[num];  /* holt Adrresse aus Screen-Struktur */
  return(plane);
 }
 /*------------------- liest die Bitmaps eines Bildes ----------------------*/
 readbody(fp)
  FILE *fp;        /* Pointer auf File-Struktur */
 {
  UWORD i,j,x,y;
  UBYTE *maps[7],*dummy;
     x = (meinbmhd.w >> 3);        /* Anzahl Bytes pro Zeile =
                                      Punkte pro Zeile / 3      */
     y = meinbmhd.h;
     for(j=0;j<meinbmhd.nPlanes;j++) {    /* hole Bitmapadrressen der maps */
  maps[j] = (UBYTE *)getbitmap((LONG)j);
     }
     for(i=0;i<y;i++) {                      /* ueber alle Zeilen */
  for(j=0;j<meinbmhd.nPlanes;j++) {                /* ueber alle Bitplanes */
      maps[j] = unpackrow(maps[j],x,fp);  /* dekomprimiere Zeile */
  }
  if(meinbmhd.masking == 1 || meinbmhd.masking == 3) { /* Maske ? */
      dummy = unpackrow(farben,x,fp);     /* dekomprimiere Maske */
                                          /* Maske wird nicht beruecksichtigt */
  }
     }
 }
 /*------------ schreibt ein Bild das im Speicher steht auf Disk -----------*/
 void savepic(name)
  char *name;      /* Name unter dem das Bild gespeichert wird */
 {
  FILE *fip;
  LONG minlen,bodylen;
  UBYTE *maps[7];
  UWORD x,y,xw;
  COUNT i,j,k;
  BYTE odd,picbuff[90];
     if((fip=fopen(name,"w"))==NULL) {
  cleanup = NULL;
  cleanexit("File kann nicht geoeffnet werden !");
     }
     cleanup = fip;
     minlen = 48L+colors;
     bodylen = 0L;
     odd = 0;
 /* minlen = 'ILBMBMHDCMAPBODY' + 12 ((lenght of chunks)*3) +
       20 (BMHD) + (length of colormap(colors))
 */
     x = (meinbmhd.w >> 3);
     y = meinbmhd.h;
```

```
/* schreibe FORM auf Disk */
    putlong(fip,IDVAL('F','O','R','M'));

/* hole Laenge des BODY-Chunks damit die Laenge des FORM-Chuncks geschrieben
   werden kann */

    for(j=0;j<meinbmhd.h;j++) {
         for(i=0;i<meinbmhd.nPlanes;i++) {
     maps[i] = s2->BitMap.Planes[i];
     bodylen += packrow(maps[i]+x*j,picbuff,(BYTE)x);
 }
    }
/* sollte die Laenge des BODY-Chuncks ungerade sein, muss ein pad-Byte ge-
   schrieben werden, da IFF-Chuncks nur gerade Laenge haben duerfen */

    odd=bodylen;
    if(odd&1) {
 putlong(fip,(minlen+bodylen+1));
    }
    else {
 putlong(fip,(minlen+bodylen));
    }
/* schreibe ILBM und BMHD auf Disk */

    putlong(fip,IDVAL('I','L','B','M'));
    meinbmhd.compression = 1;
    meinbmhd.masking = 0;
    putlong(fip,IDVAL('B','M','H','D'));
    putlong(fip,20L);
    loc_bmhd = (UBYTE *)(&meinbmhd);
    for(i=0;i<20;i++) {
 putbyte(fip,*loc_bmhd++);
    }
/* schreibe den CMAP-Chunck auf Disk */

    putlong(fip,IDVAL('C','M','A','P'));
    putlong(fip,colors);
    loc_cmap = farben;
    for(i=0;i<colors;i++) {
 putbyte(fip,*loc_cmap++);
    }
/* schreibe BODY-Chunck
   das Bild wird immer komprimiert und ohne Maske abgespeichert */

    putlong(fip,IDVAL('B','O','D','Y'));
    putlong(fip,bodylen);
    for(j=0;j<meinbmhd.h;j++) {
         for(i=0;i<meinbmhd.nPlanes;i++) {
     maps[i] = s2->BitMap.Planes[i];
     bodylen = packrow(maps[i]+x*j,picbuff,(BYTE)x);
     for(k=0;k<bodylen;k++) {
         putbyte(fip,picbuff[k]);
     }
 }
    }

    if(odd) {    /* ungerade, dann schreibe pad-Byte */
 putbyte(fip,0);
    }

    fclose(fip);
    cleanup = NULL;

}


/*------- erstellt eine Rotationsmatrix mit den Winkeln x,y, und z ----------*/

void makerotmatrix(x,y,z)

 double x,y,z;

{

    m[0][0] =  cos(y)*cos(z);
    m[0][1] = -cos(y)*sin(z);
    m[0][2] =  sin(y);
    m[1][0] =  cos(x)*sin(z) + sin(x)*sin(y)*cos(z);
    m[1][1] =  cos(x)*cos(z) - sin(x)*sin(y)*sin(z);
    m[1][2] = -sin(x)*cos(y);
    m[2][0] =  sin(x)*sin(z) - cos(x)*sin(y)*cos(z);
    m[2][1] =  sin(x)*cos(z) + cos(x)*sin(y)*sin(z);
    m[2][2] =  cos(x)*cos(y);

}


/*------- multipliziert einen Vektor mit der Matrix m ---------------------*/

void rotatevec(v1,ve)

 double *v1,*ve; /* v1 = Pointer auf den zu multiplizierenden Vektor */
                 /* ve = Pointer auf Vektor in dem das Ergebnis steht */

{

 COUNT i,j;
 double h;

    for(i=0;i<3;i++) {
 h = 0.0;
 for(j=0;j<3;j++) {
     h += (m[i][j] * v1[j]);
 }
 ve[i] = h;
    }

}


/*---------- Berechnung der y-Groesse fuer gleiche Breite und Hoehe ---------*/

double getyprop(xs,xw,yw)

 double xs,xw,yw;

{

 double ys,xh;

    xh = xs * BIGESTWIN(xw/2.0);
    ys = xh/BIGESTWIN(yw/2.0);

/* Der Wert "MONITOR" kann veraendert werden, und haengt von der Einstellung
   Ihres Monitors ab. Der Wert gibt an, um wieviel die einzelnen Pixel
   breiter als hoch sind; er muss im Zweifelsfall durch probieren er-
   mittelt werden. */

    /* vermeide Division durch Null */
    ys = (double)(ys == 0.0) ? (1.0) : (ys);
    return(ys);

}


/*---------- Berechnung der x-Groesse fuer gleiche Breite und Hoehe ---------*/

double getxprop(ys,xw,yw)

 double ys,xw,yw;

{

 double xs,yh;

    yh = ys * BIGESTWIN(yw/2.0);
    xs = yh/BIGESTWIN(xw/2.0);

    /* vermeide Division durch Null */
    xs = (double)(xs == 0.0) ? (1.0) : (xs);

    return(xs);

}


/*---------- Transformation eines Bildes -----------------------------------*/

void transformpic(src,dest,xa,xe,ya,ye,xwink,ywink,
        xoff,yoff,xsize,zsize,ref,xrot,yrot,zrot,mode)

 struct RastPort *src,*dest;
 LONG xa,xe,ya,ye;
 double xwink,ywink;
 LONG xoff,yoff;
 double xsize,zsize;
 BOOL ref;
 double xrot,yrot,zrot;
 LONG mode;

{

 double xstep,zstep;
 double xstart,ystart,xstop,ystop;
 double xpic,ypic,ysize;
 double pixstep,piystep;
 double x,z;
 double k[3],h[3],gs[2];
 LONG color,xclip,yclip;

    ScreenToFront(s2);   /* zeige Screen mit Transformierten Bild */

    makerotmatrix(TOBOGEN(xrot),TOBOGEN(yrot),TOBOGEN(90.0+zrot));

    ysize = xsize;
    ystart = -TOBOGEN(ywink/2.0);
    ystop  = TOBOGEN(ywink/2.0);
    xstart = -TOBOGEN(xwink/2.0);
    xstop  = TOBOGEN(xwink/2.0);
    xstep = (MAXWIN(zsize));
    zstep = (MAXWIN(xsize));
    pixstep = ((double)(xe-xa))/(TOBOGEN(xwink)/zstep);
    piystep = ((double)(ye-ya))/(TOBOGEN(ywink)/xstep);

```

```
    SetRast(dest,0L);     /* loesche Screen Nr. 2 */

    ypic = (double)ya;

    for(x=ystart ;x<ystop ;x+=xstep) {    /* ueber alle Zeilen (y) */
 xpic = (double)xa;                  /* setze x auf 0 */

 for(z=xstart; z<xstop;z+=zstep) {        /* ueber alle Spalten (x) */

     color = ReadPixel(src,xa+(LONG)(xpic),ya+(LONG)(ypic));
     if(color != 0L) {              /* wenn kein Punkt, weiter */

         /* errechne die Koordinaten der Kugeloberflaeche */
         h[0] = xsize*cos(x)*cos(z);
         h[1] = ysize*cos(x)*sin(z);
         h[2] = zsize*sin(x);
         rotatevec(h,k);            /* rotiere Punkt der Oberflaeche */

         SetAPen(dest,color);     /* uebertrage Farbe */
         switch(ref) {
             case(0):{
                 xclip = xoff-(LONG)k[0];
                 yclip = yoff+(LONG)(k[2]*ASPECT);
                 break;
             }
             case(1):{
                 xclip = xoff-(LONG)(k[0]/ASPECT);
                 yclip = yoff+(LONG)k[2];
                 break;
             }
             default:{
                 xclip = xoff-(LONG)k[0];
                 yclip = yoff+(LONG)k[2];
                 break;
             }
         }
         xclip = ((xclip<0L)||(xclip>=(LONG)meinbmhd.w)) ? -1L : xclip;
         yclip = ((yclip<0L)||(yclip>=(LONG)meinbmhd.h)) ? -1L : yclip;

         if((xclip >= 0) && (yclip >= 0)) {
             switch(mode) {

                 case VOLL :{    /* undurchsichtig */
                     if(k[1]>0.0) {      /* zeigt Vektor nach vorn ? */
                         WritePixel(dest,xclip,yclip);
                     }
                     break;
                 }

                 case LEER :{
                     if(k[1]<0.0) {      /* zeigt Vektor nach hinten ? */
                         /* dann setze Pixel wenn noch keiner ge-
                            setzt ist */

                         if(ReadPixel(dest,xclip,yclip)==0L)
                             WritePixel(dest,xclip,yclip);
                     }
                     else {
                         WritePixel(dest,xclip,yclip);
                     }
                     break;
                 }

             }  /* ende switch */

         }      /* ende clipping */

     } /* ist keine Farbe gesetzt, wird hier weitergemacht  */

     /* naechste x-position */
     xpic += pixstep;

 } /* ende ueber alle Spalten */

 if(s2->TopEdge >= 230) {        /* Screen Nr.2 unten ? */
     WBenchToFront();
     MoveScreen(s2,0L,-512L);
     printf("IFF-Transformation unterbrochen.\n");
     cont:
     printf("Weiter (j,n) ? ");
     scanf("%s",zeichenpuffer);
     if(zeichenpuffer[0]=='n') {
         printf("IFF-Transformation abgebrochen .\n");
         return();
         }
     else {
         if(zeichenpuffer[0]!='j')
         goto cont;
     }
     WBenchToBack();
 }

 /* naechste y-Position */
 ypic += piystep;

    }  /* ende ueber alle Zeilen */

    printf("IFF-Transformation beendet.\n");

}


/*----------------------------------------------------------------------*/

main()

{

    GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0L);
    if(GfxBase == NULL)
 cleanexit("graphics library kann nicht geoeffnet werden !");

    IntuitionBase = (struct IntuitionBase *)OpenLibrary("intuition.library",0L);
    if(IntuitionBase == NULL)
 cleanexit("intuition library kann nicht geoeffnet werden !",NULL);

    printf("\n\n        IFF-Transformer V1.3  \n");
    printf(" written in 1987 by Christian Hoench.\n\n");
    printf("Name des zu transformierenden Bildes => ");
    scanf("%s",zeichenpuffer);
    printf("loading %s\n",zeichenpuffer);
    loadpic(zeichenpuffer);
    WBenchToFront();
    dotrans();
    getuser:
    WBenchToFront();
    printf("Speichern, Neustart, Laden oder Ende (s,n,l,e) ? ");
    scanf("%s",zeichenpuffer);
    if(zeichenpuffer[0]=='e') {
 cleanexit("IFF-Transformation beendet.\n");
    }
    if(zeichenpuffer[0]=='n') {
 dotrans();
 goto getuser;
    }
    if(zeichenpuffer[0]=='s') {
 name:
 printf("Name des Bildes => ");
 scanf("%s",zeichenpuffer);
 printf("saving %s.\n",zeichenpuffer);
 savepic(zeichenpuffer);
    }
    if(zeichenpuffer[0]=='l') {
 printf("Name des zu transformierenden Bildes => ");
 scanf("%s",zeichenpuffer);
 printf("loading %s\n",zeichenpuffer);
 loadpic(zeichenpuffer);
    }
    goto getuser;
}


/*--------------- hole Werte von User, starte Transformation ---------------*/

dotrans()
{
 WORD xan,xen,yan,yen,xo,yo,mo;
 double xwin,ywin,xr,yr,zr,xsi,ysi;
 BOOL xprop,yprop,ref;

    ref = 2;
    printf("Parameter eingeben:\n");

    xa:
    printf("xstartpositon (0-%d) ? ",meinbmhd.w);
    scanf("%d",&xan);
    if(xan<0 || xan>meinbmhd.w) {
 printf("illegaler Wert.\n");
 goto xa;
    }

    xe:
    printf("xstopposition (0-%d) ? ",meinbmhd.w);
    scanf("%d",&xen);
    if(xen<0 || xen>meinbmhd.w) {
 printf("illegaler Wert.\n");
 goto xe;
    }

    ya:
    printf("ystartpositon (0-%d) ? ",meinbmhd.h);
    scanf("%d",&yan);
    if(yan<0 || yan>meinbmhd.h) {
 printf("illegaler Wert.\n");
 goto ya;
    }

    ye:
    printf("ystopposition (0-%d) ? ",meinbmhd.h);
    scanf("%d",&yen);
    if(yen<0 || yen>meinbmhd.h) {
 printf("illegaler Wert.\n");
 goto ye;
    }
```

```
    xwin:
    printf("xwinkel (1-360) ? ");
    scanf("%f",&xwin);
    if((xwin>360.0) || (xwin<1.0)) {
printf("illegaler Wert.\n");
goto xwin;
    }

    ywin:
    printf("ywinkel (1-180) ? ");
    scanf("%f",&ywin);
    if((ywin>180.0) || (ywin<1.0)){
printf("illegaler Wert.\n");
goto ywin;
    }

    rot:
    printf("xrot,yrot,zrot ? ");
    scanf("%f,%f,%f",&xr,&yr,&zr);

    xo:
    printf("xoffset,yoffset ? ");
    scanf("%d,%d",&xo,&yo);
    if((xo<0 || xo>meinbmhd.w) || (yo<0 || yo>meinbmhd.h)) {
printf("illegaler Wert.\n");
goto xo;
    }

    six:
    xprop = FALSE;
    printf("xradius (0 = X-Propmod.) ? ");
    scanf("%f",&xsi);
    if(xsi < 1.0) {
if(xsi == 0.0) {
     xprop = TRUE;
     printf("  X-Propotionalmodus eingeschaltet.\n");
}
else {
     printf("illegaler Wert.\n");
     goto six;
}
    }

    siy:
    yprop=FALSE;
    printf("yradius (0 = Y-Propmod.) ? ",meinbmhd.h/2);
    scanf("%f",&ysi);
    if(ysi < 1.0) {
if(ysi == 0.0) {
     if(xprop) {
         printf("Kein xradius vorhanden (X-Proportionalmodus) !\n");
         goto siy;
     }
     ysi = getyprop(xsi,xwin,ywin);
     printf("  Y-Proportionalmodus eingeschaltet.\n");
     printf("  yradius = %f\n",ysi);
     yprop=TRUE;
     goto rg;
}
printf("illegaler Wert.\n");
goto siy;
    }
    if(xprop) {
xsi = getxprop(ysi,xwin,ywin);
printf("  xradius = %f\n",xsi);
    }

    rg:
    if((ASPECT != 1.0) && (xprop || yprop)) {
printf("  Referenzgroesse eingeben (x oder y) ? ");
scanf("%s",zeichenpuffer);
if(zeichenpuffer[0]=='x') {
     printf("  xradius ist Referenzgroesse.\n");
     ref = 0;
}
else {
     if(zeichenpuffer[0]!='y') {
         goto rg;
     }
     printf("  yradius ist Referenzgroesse.\n");
     ref = 1;
}
    }

    mo:
    if((xsi > meinbmhd.w/2) || (ysi > meinbmhd.h/2)) {
 if((xwin < 180.0) || (ywin < 180.0))
     printf("  Achtung: Teile koennen unsichtbar sein !\n");
    }
    printf("Modus (0 or 1) ? ");
    scanf("%d",&mo);
    if(mo!=0 && mo!=1) {
 printf("  unbekannter Modus.\n");
 goto mo;
    }
    WBenchToBack();
    transformpic(rp,rp2,
 (LONG)xan,(LONG)xen,(LONG)yan,(LONG)yen,
 xwin,ywin,
 (LONG)xo,(LONG)yo,
 xsi,ysi,ref,xr,yr,zr,
 (LONG)mo);

}


/*---- Verlassen des Programms und schliessen der Screen,Libraries,Files ----*/

cleanexit(errmes)

 char* errmes;

{
 int i;

    if(errmes != NULL) {
 printf("%s\n",errmes);
    }
    if(cleanup != NULL) {
 fclose(cleanup);
    }
    if(s != NULL) {
 CloseScreen(s);
    }
    if(s2 != NULL) {
 CloseScreen(s2);
    }
    if(IntuitionBase != NULL) {
 CloseLibrary(IntuitionBase);
    }
    if(GfxBase != NULL) {
 CloseLibrary(GfxBase);
    }

    exit(0L);

}
```

VON OLIVER EDLER

WANDLER ZWISCHEN DEN WELTEN

OBJEKTKONVERTIERUNG VON SCULPT3D ZU VIDEOSCAPE3D

Jeder, der schon einmal versucht hat, per Millimeterpapier und Bleistift ein Objekt für Videoscape zu definieren, und kaum über den obligatorischen Würfel hinausgekommen ist, wird sich schon nach 3dimensionalem Papier mit eingebautem Texteditor gesehnt haben, denn spätestens nach dem 20. Punkt und der 40.Fläche sieht das Millimeterpapie eher wie ein Schnittmuster aus. Doch wofür gibt es Computer und Programme wie Sculpt 3-D? Leider sieht Videoscape es nicht vor, Daten im IFF-Format, das es inzwischen auch schon für Objekte gibt, einzulesen. Damit Sie dennoch zu Ihrem Raumschiff Enterprise kommen, hier nun die Beschreibung unseres Programmes. Eine Erklärung des IFF-Datenformates befindet sich in Form von REM-Zeilen, die natürlich nicht mit abgetippt werden müssen, im Listing.

Das PROJECT- Menue: Der Menü-Punkt SCULPT LOAD gibt Ihnen die Möglichkeit, entweder eine komplette Sculpt-Scene zu laden, um sich zusätzlich einen Überblick über den Beobachterstandort, evtl. vorhandene Lampen, eine Liste aller Kanten und die Umgebung des Objektes zu verschaffen , oder Sie laden nur die reinen Objekt-Daten wie Eckpunkte und eine Flächenverbindungsliste. Dabei spielt es keine Rolle, ob Ihre Scene aus mehreren Objekten besteht. Das Programm macht aus ihnen ein einziges Videoscape-Objekt. Mit VSCAPE SAVE speichern Sie das konvertierte Videoscape-Objekt ab. Die Farbdaten werden nicht automatisch umgesetzt, sondern durch einen Defaultwert (VSdef) global gesetzt. Allerdings kann dieser Farbwert im VOREINSTELLUNG-Menü geändert werden. Falls Ihr Sculpt-Objekt mehrere Farben enthält, können Sie diese im BEARBEITEN-Menü einzeln edieren. Hierbei wird die Sculpt-Oberflächenstruktur automatisch umgerechnet. ENDE braucht wohl nicht näher erläutert werden, außer daß Sie sichergehen sollten, Ihr modifiziertes Objekt auch abge-

speichert zu haben. Hierfür ist aber noch einmal eine Sicherheitsabfrage eingebaut.

Das DATEN ZEIGEN- Menü:

Je nachdem, was Sie vorher geladen haben, können Sie sich jetzt die Daten des Sculpt-Objektes ansehen oder ausdrucken lassen. Versierte wissen sicher etwas mit Standort, Blickpunkt, Lampenposition usw. anzufangen. Zum Beispiel verwenden Sie den Standort (Observer) in Ihrem CAM-File als Kamera-Position. Die Blickpunkt-Daten (Target) geben den Koordinatenursprung an. Außerdem werden es einige nützlich finden, die ausgedruckten Daten zur Archivierung der Sculpt-Objekte zu verwenden.

Das BEARBEITEN-Menü:

Bei der Option FARBEN werden Ihnen die zur Verfügung stehenden Videoscape-Farben im oberen Fensterteil angezeigt. Darunter sehen Sie die Farbe der momentan zu bearbeitenden Fläche des Sculpt-Objekts. Sie müssen nun aus der Palette die neue Farbe auswählen. Doch Vorsicht ! Durch nochmaliges Anwählen des VOREINSTELLUNG-Menüs werden alle Flächen wieder einheitlich gefärbt. ACHSEN gibt Ihnen die Möglichkeit, die Lage Ihres Objektes zu verändern. Das Programm fragt Sie, welche beiden Achsen vertauscht werden sollen. Menüpunkt DIMENSION: Damit Ihr VideoScape-Object nicht zu groß wird, können Sie hier die Dimension der Koordinatenwerte einstellen. ANZEIGEN ist eine 'Minimalfunktion', um Ihr Objekt, um einen festen Winkel von 45 Grad gedreht, zu zeichnen. Auf die Daten wird hierbei kein Einfluß ausgeübt. Jede Aktion kann entweder durch den Menüpunkt ABBRECHEN oder CTRL-C abgebrochen werden.

```
REM IFF-Format sculpt scene
REM
REM HEADER
REM FORM-Chunk:
REM 4 bytes        "FORM"
REM long           'Gesamtlaenge in bytes
REM Chunk-Art      "SC3D"
REM
REM LAMP-Chunk:
REM 4 bytes        "LAMP"
REM long           'chunklaenge in bytes
REM long           'X-lampe
REM long           'Z-lampe
REM long           'Y-lampe
REM long           'helligkeit
REM 3 bytes        'r-g-b lampenfarbe
REM 1 byte          'fuellbyte
REM
REM OBSV-Chunk:
REM 4 bytes        "OBSV"
REM long           'chunklaenge in bytes
REM long           'mode 0=paint,1=snapshot,2=photo,3=wire
frame
REM long           'lens Brennweite
REM long           '? suspect 100-150
REM long           '? suspect 4-6
REM long           'X-location
REM long           'Y-location
REM long           'Z-location
REM long           'X-target
REM long           'Y-target
REM long           'Z-target
REM word           'resolution 0=lo-res,1=hi-res
REM word           '0=no-interlace,1=interlace
REM word           'lens 0=normal,1=wideangle,2=telephoto,
3=special
REM word           'exposure 0=auto,1=manual
REM long           'lens special
REM long           'exposure prozent
REM long           'exposure manual
REM long           'imagesize 0=tiny,1=small,2=medium,3=fu
ll,4=jumbo
REM long           'angle of tilt
REM long           'anti-alias 0=none,1=good,2=best
REM
REM WRLD-Chunk:
REM 4 bytes        "WRLD"
REM long           'groundmode 0=none,1=solid,2=checkered
REM long           'skymode     0=none,1=solid,2=graduated
REM long           'dimension fuer ground-raster
```

```
REM 3 bytes      'r-g-b illumination
REM 3 bytes      'r-g-b ground1
REM 3 bytes      'r-g-b ground2
REM 3 bytes      'r-g-b sky1
REM 3 bytes      'r-g-b sky2
REM
REM VERT-Chunk:
REM 4 bytes      "VERT"
REM long         'chunklaenge in bytes
REM long         'X-eckpunkt
REM long         'Z-eckpunkt
REM long         'Y-eckpunkt
REM
REM EDGE-Chunk:
REM 4 bytes      "EDGE"
REM long         'chunklaenge in bytes
REM long         'von Punkt Nr.
REM long         'bis Punkt Nr.
REM
REM FACE-Chunk:
REM 4 bytes      "FACE"
REM long         'chunklaenge in bytes
REM long         'von Punkt Nr.
REM long         'bis Punkt Nr.
REM long         'bis Punkt Nr.
REM 3 bytes      'r-g-b Flaeche
REM byte         'texture 0=dull,1=shiny,2=mirror,3=lumi
nous,4=glass,bit#8=smoothing
REM
REM
CLEAR ,100000&
WINDOW 2,"                                   METAMORPH",,
0
REM
REM Menue vorbereiten
REM
MENU 1,0,1," PROJECT"
MENU 1,1,1,"SCULPT LOAD "
MENU 1,2,1,"VSCAPE SAVE "
MENU 1,3,1,"ENDE        "

MENU 2,0,1,"DATEN ZEIGEN"
MENU 2,1,0,"UMGEBUNG    "
MENU 2,2,0,"LAMPEN      "
MENU 2,3,0,"BEOBACHTER  "
MENU 2,4,1,"ECKEN       "
MENU 2,5,1,"FLÄCHEN     "
MENU 2,6,0,"KANTEN      "
MENU 2,7,0,"ABBRECHEN   "

MENU 3,0,1,"BEARBEITEN  "
MENU 3,1,1,"FARBEN      "
MENU 3,2,1,"ACHSEN      "
MENU 3,3,1,"DIMENSION   "
MENU 3,4,1,"ANZEIGEN    "
MENU 3,5,0,"ABBRECHEN   "

MENU 4,0,1,"VOREINSTELLUNG"
MENU 4,1,1,"FARBEN        "

REM
REM Unterbrechungsereignisse bestimmen
REM
ON MENU GOSUB AUSWAHL
MENU ON
ON BREAK GOSUB Ende
BREAK ON
ON ERROR GOTO Fehler
REM
VSdef=15     'Defaultwert fuer VideoScape-Farbe
DIM Vcol(15),Vcol$(16)'Array zur Umrechnung colornr. ->
 Vscapenr.
DIM Fl(200,8),Ecke(100,3),Kante(200,2),Lampe(5,7)
RESTORE VSCAPEFARBEN
FOR x=4 TO 14
  READ Vcol(x)
NEXT x
```

```
FOR x=0 TO 15
  READ Vcol$(x)
NEXT
FOR x=0 TO 3
  READ vmode$(x)
NEXT
VSCAPEFARBEN:
DATA 0,1,2,4,6,7,9,10,12,14,15
DATA black,"dark blue","dark green",(unused),"dark red"
,(unused)
DATA brown,gray,black,"light blue","light green",(unuse
d),"light red",(unused),yellow,white
DATA matte,glossy,unshaded,"unfilled outline"
REM
GOSUB Laden
BEEP
REM
REM Programm ist voellig unterbrechungsgesteuert
REM
main:
  WHILE 1
    SLEEP
  WEND
REM
Fehler:
CLS
IF ERR =53 THEN  'File not found
  CLS
  LOCATE 3,25:PRINT "Object ";Dl$;".scene nicht gefunde
n !"
  FOR I=0 TO 5000:NEXT
  RESUME Laden
END IF
ERROR ERR
Ende:
  a$=""
  a=CSRLIN         'Cursorposition merken
  LOCATE 1,1:PRINT "WOLLEN SIE DAS PROGRAMM WIRKLICH BE
ENDEN ?"
  WHILE a$<>"J"AND a$<>"N"
    a$=UCASE$(INKEY$)
  WEND
  IF a$="N" THEN  'Cursor an alte position und weiter
    LOCATE a,1:PRINT "
    "
    RETURN
  END IF
  CLS
  SCREEN CLOSE 1
  MENU RESET
  CLOSE
  WINDOW CLOSE 2
  END
REM
REM Unterprogramm Menue-Nr.
REM
AUSWAHL:
  m%=MENU(0) 'NR. des gewaehlten Menues
  P%=MENU(1) 'Menuepunkt NR.
  ON m% GOSUB DATEI,DATEN,BEARBEITEN,VOREIN
  m%=0:P%=0
  BEEP
RETURN
REM
REM Unterprogramme Menue-Punkt
REM
DATEI:
  ON P% GOSUB Laden,Speichern,Ende
  CLS
RETURN
DATEN:
  CLS
  GOSUB Ausgabe
  OPEN device$ FOR OUTPUT AS 1
  ON P% GOSUB WRLD,LAMP,OBSV,VERT,FACE,EDGE
  MENU 2,7,0
  CLOSE
```

INFO

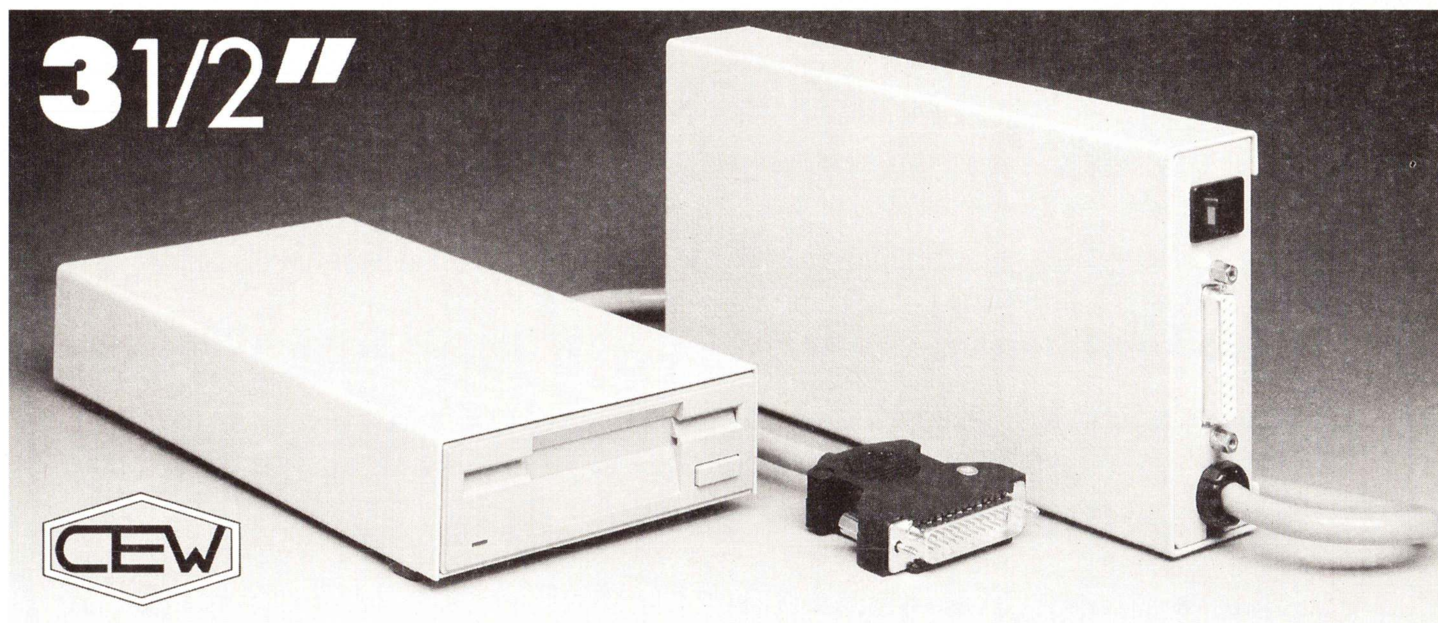


Qualitäts-Hardware

Made in Germany

3½"-Amiga-Laufwerk extern

- Industriestandard
- 880 KB Speicherkapazität
- EIN/AUS Schalter
- Durchgeführter Port mit Schraubverriegelungen
- Solides Metallgehäuse



Weitere Produkte aus unserem Hause:

- 5¼"-Amiga-Zusatzlaufwerk Extern
- 3½"-Amiga-Zusatzlaufwerk Intern
- 3½"-Atari-Zusatzlaufwerk Extern
- 512 KB Speichererweiterung für Amiga 500 mit Uhr
- 5¼"-Zusatzlaufwerk Extern z. B. für Mitsubishi Laptop
- Computerleitungen für alle gängigen Computertypen
- 3½" Discdrive für XT/AT
- 3½" und 5¼" Discdrives
- Midi-Interface für Amiga
- Sound-Digitizer Stereo für Amiga

Erhältlich bei:

VOBIS DATA COMPUTER GMBH

HAKO® FOTO · VIDEO · ELEKTRONIK

Rainbow Data

und im gut sortierten Fachhandel.

Mettmanner Straße 66
5620 Velbert 1
Telefon (0 20 51) 5 92 97
Telex 859 766 5 CEW
Telefax (0 20 51) 5 90 32

```
RETURN
BEARBEITEN:
  ON P% GOSUB Farben,ACHSEN,Dimension,Anzeige
  MENU 3,5,0
RETURN
VOREIN:
ON P% GOSUB Default
RETURN
REM
REM Laderoutine
REM
Laden:
  CLS
  sel$=""
  Teiler=10
  LOCATE 3,10:INPUT"BITTE NAMEN DER ZU LESENDEN SCULPT
DATEI EINGEBEN:";Dl$
  Dl$=UCASE$(Dl$)
  REM
  REM Filename soll ohne appendix '.scene' eingegeben w
erden
  REM
  IF Dl$="" THEN Laden
  I=INSTR(Dl$,".SCENE")
  IF I<>0 THEN
   Dl$=MID$ (Dl$,1,I-1):
  END IF
  REM
  LOCATE 6,20:PRINT "Wollen Sie das gesamte file einles
en,oder nur"
  LOCATE 8,20:PRINT "die für VideoScape relevanten Obje
kt-Daten ?"
  LOCATE 10,29:PRINT "(G)esamt / (O)bject"
  WHILE sel$<>"G"AND sel$<>"O"
    sel$=UCASE$(INKEY$)
    WEND
  OPEN Dl$+".SCENE" FOR INPUT AS 1
  FormId$=INPUT$(4,1)           'IFF-ID einlesen
  IF FormId$<>"FORM" THEN
    CLS :BEEP:BEEP
    PRINT "KEINE IFF DATEI"
    FOR I=1 TO 10000:NEXT I
    CLOSE 1
    RETURN
  END IF
  formlaenge=CVL(INPUT$(4,1))
  formkennung$=INPUT$(4,1)      'Formname = 'SC3D' ?
  IF formkennung$<>"SC3D" THEN
    CLS :BEEP:BEEP
    PRINT "KEINE SCULPT DATEI"
    FOR I=1 TO 10000:NEXT I
    CLOSE 1
    RETURN
  END IF
  CLS
  REM
  REM Wenn nur Object eingelesen,dann
  REM Menuepunkte in Geisterschrift
  REM
  Flag=ABS(sel$="G")
  MENU 2,1,Flag
  MENU 2,2,Flag
  MENU 2,3,Flag
  MENU 2,6,Flag
REM
REM Chunkweises einlesen der Daten
REM
CHUNKLESEN:
  IF EOF(1) THEN CLOSE#1:RETURN
  chunkname$ =INPUT$(4,1)
  chunklaenge =CVL(INPUT$(4,1))
  IF sel$="G" THEN
    IF chunkname$="LAMP" THEN  LAMPEN
    IF chunkname$="OBSV" THEN  BEOBACHTER
    IF chunkname$="WRLD" THEN  WELT
    IF chunkname$="EDGE" THEN  KANTEN
  END IF
```

```
  IF chunkname$="FACE" THEN  FLAECHE
  IF chunkname$="VERT" THEN  ECKPUNKTE
  rest$=INPUT$(chunklaenge,1):rest$=""
GOTO CHUNKLESEN
RETURN

ECKPUNKTE:
  Max=0:Min=0
  Anzpunkte = chunklaenge/4/3
  IF UBOUND (Ecke)<Anzpunkte THEN
    ERASE Ecke
    DIM Ecke(Anzpunkte,3)
  END IF
  LOCATE 2,3:PRINT "LESE ECKPUNKT NR."
  FOR x = 0 TO Anzpunkte-1
    LOCATE 2,19:PRINT x
    Ecke(x,0)=CVL(INPUT$(4,1))
    Ecke(x,2)=CVL(INPUT$(4,1))
    Ecke(x,1)=CVL(INPUT$(4,1))
    REM
    REM Maximal und Minimalwert der Koordinaten
    REM werden gleich bei Einlesen ermittelt
    REM
    IF Max<Ecke(x,0) THEN Max=Ecke(x,0)
    IF Max<Ecke(x,1) THEN Max=Ecke(x,1)
    IF Max<Ecke(x,2) THEN Max=Ecke(x,2)
    IF Min>Ecke(x,0) THEN Min=Ecke(x,0)
    IF Min>Ecke(x,1) THEN Min=Ecke(x,1)
    IF Min>Ecke(x,2) THEN Min=Ecke(x,2)
  NEXT
  REM
  REM Optimale VScape Dimension errechnen
  REM
  Mittel = Max+ABS(Min)/2
  WHILE Mittel>100
    Mittel = Mittel/Teiler
    Teiler = Teiler*10
  WEND
GOTO CHUNKLESEN

KANTEN:
  LOCATE 6,3:PRINT "LESE KANTE NR."
  Anzkant=chunklaenge/4/2
  IF UBOUND (Kante)<Anzkant THEN
    ERASE Kante
    DIM Kante(Anzkant,2)
  END IF
  FOR x = 0 TO Anzkant-1
    LOCATE 6,19:PRINT x+1
    Kante(x,0) = CVL(INPUT$(4,1))
    Kante(x,1) = CVL(INPUT$(4,1))
  NEXT
GOTO CHUNKLESEN

FLAECHE:
  LOCATE 4,3:PRINT "LESE FLÄCHE NR."
  Anzflach = chunklaenge/4/4
  IF UBOUND (Fl)<Anzflach THEN
    ERASE Fl
    DIM Fl(Anzflach,8)
  END IF
  FOR x = 0 TO Anzflach-1
    LOCATE 4,19:PRINT x+1
    Fl(x,0) = CVL(INPUT$(4,1))
    Fl(x,1) = CVL(INPUT$(4,1))
    Fl(x,2) = CVL(INPUT$(4,1))
    Fl(x,3) = ASC(INPUT$(1,1))
    Fl(x,4) = ASC(INPUT$(1,1))
    Fl(x,5) = ASC(INPUT$(1,1))
    Fl(x,6) = ASC(INPUT$(1,1))
    Fl(x,7) = VSdef  'defaultwert zuweisen
  NEXT x
GOTO CHUNKLESEN

LAMPEN:
  LOCATE 2,3:PRINT "LESE LAMPENDATEN"
```

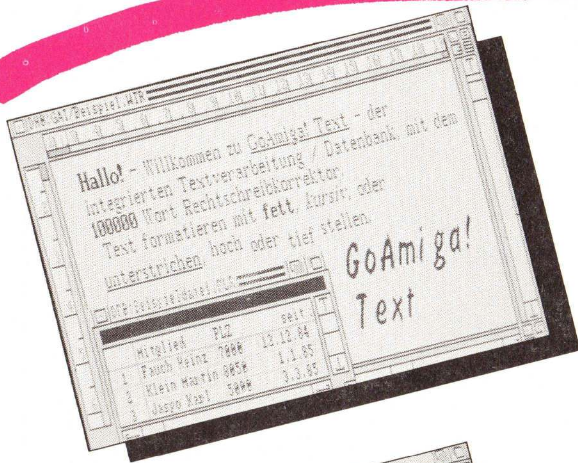
```
   Anzlamp = chunklaenge/4/5
   IF UBOUND (Lampe)<Anzlamp THEN
     ERASE Lampe
     DIM Lampe(Anzlamp,7)
   END IF
   FOR x=1 TO Anzlamp
     Lampe(x,0) = CVL(INPUT$(4,1))
     Lampe(x,2) = CVL(INPUT$(4,1))
     Lampe(x,1) = CVL(INPUT$(4,1))
     Lampe(x,3) = CVL(INPUT$(4,1))
     Lampe(x,4) = ASC(INPUT$(1,1))
     Lampe(x,5) = ASC(INPUT$(1,1))
     Lampe(x,6) = ASC(INPUT$(1,1))
     dummy = ASC(INPUT$(1,1))
   NEXT x
GOTO CHUNKLESEN

BEOBACHTER:
   LOCATE 4,3:PRINT "LESE BEOBACHTER"
   mode   = CVL(INPUT$(4,1))
   lens   = CVL(INPUT$(4,1))
   susp   = CVL(INPUT$(4,1))
   susp   = CVL(INPUT$(4,1))
   xloc   = CVL(INPUT$(4,1))
   zloc   = CVL(INPUT$(4,1))
   yloc   = CVL(INPUT$(4,1))
   xtar   = CVL(INPUT$(4,1))
   ztar   = CVL(INPUT$(4,1))
   ytar   = CVL(INPUT$(4,1))
   res    = CVI(INPUT$(2,1))
   inter  = CVI(INPUT$(2,1))
   lmode  = CVI(INPUT$(2,1))
   exmode= CVI(INPUT$(2,1))
   lens2  = CVL(INPUT$(4,1))
   expo1  = CVL(INPUT$(4,1))
   expo2  = CVL(INPUT$(4,1))
   size   = CVL(INPUT$(4,1))
   tilt   = CVL(INPUT$(4,1))
   alias  = CVL(INPUT$(4,1))
   rest$  = INPUT$(chunklaenge-72,1)
   rest$  = ""
GOTO CHUNKLESEN

WELT:
   LOCATE 6,3:PRINT "LESE UMGEBUNGSDATEN"
   groundmode= CVL(INPUT$(4,1))
   skymode    = CVL(INPUT$(4,1))
   Dimension = CVL(INPUT$(4,1))
   Rilu = ASC(INPUT$(1,1))
   Gilu = ASC(INPUT$(1,1))
   Bilu = ASC(INPUT$(1,1))
   Rgr1 = ASC(INPUT$(1,1))
   Ggr1 = ASC(INPUT$(1,1))
   Bgr1 = ASC(INPUT$(1,1))
   Rgr2 = ASC(INPUT$(1,1))
   Ggr2 = ASC(INPUT$(1,1))
   Bgr2 = ASC(INPUT$(1,1))
   Rsk1 = ASC(INPUT$(1,1))
   Gsk1 = ASC(INPUT$(1,1))
   Bsk1 = ASC(INPUT$(1,1))
   Rsk2 = ASC(INPUT$(1,1))
   Gsk2 = ASC(INPUT$(1,1))
   Bsk2 = ASC(INPUT$(1,1))
   rest$= INPUT$(chunklaenge-27,1)
   CLS
GOTO CHUNKLESEN

Speichern:
   CLS
   LOCATE 2,1
   INPUT "BITTE NAMEN DES ZU SPEICHERNDEN VIDEOSCAPE OBJ
ECTS ANGEBEN :";DS$
   IF DS$="" THEN DS$=D1$
   OPEN DS$+".geo" FOR OUTPUT AS #1
   PRINT#1,"3DG1"
   PRINT#1,Anzpunkte
   FOR x=0 TO Anzpunkte-1
```

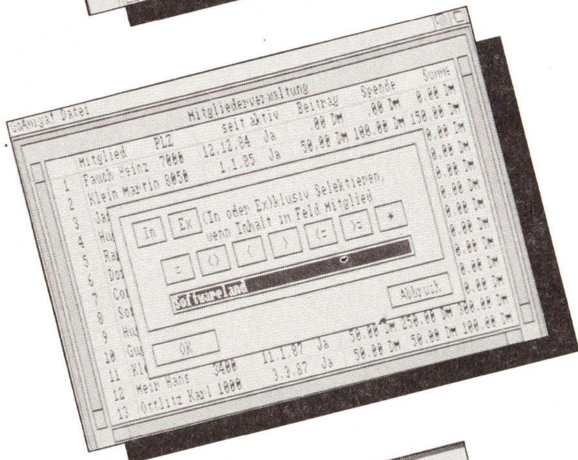
Die Besten!



Die Textverarbeitung

GoAmiga! Text ist sowohl Textverarbeitung als auch Dateiverwaltung in einem Paket. Dadurch wird das Erstellen von Listen zum Kinderspiel. Ein integrierter Rechtschreibkorrektor sorgt für die notwendige Genauigkeit. Höchster Bedienungskomfort. Lieferbar ab ca. März 88.

GOAMIGA! TEXT
DM 298.00 SFr. 298.00 ÖS 2652.00



Die Datenbank

GoAmiga! Datei ist die einzige Datenbank die es erlaubt, neben Text- und Bilddaten auch Tonfolgen abzuspeichern und zu verarbeiten. Die Übernahme von Daten aus anderen Programmen ist mittels ASCII-Umlaut-Schnittstelle problemlos möglich. Mitgeliefert wird ein englisch-deutsches Wörterbuch.

GOAMIGA! DATEI
DM 199.00 SFr. 199.00 ÖS 1770.00



Die Titelsoftware

GoAmiga! Titel ist ein nützliches und flexibles Instrument zur Herstellung von Titeln, Laufschriften, Bild- und Toneffekten. Die Vorspänne können auf Video aufgezeichnet werden oder aber mittels eines Abspielprogramms in den Anfang einer beliebigen Diskette eingebunden werden.

GOAMIGA! TITEL
DM 89.00 SFr. 89.00 ÖS 791.00

Bestellservice

BRD: 089-8340591
G. Lechner
Planeggerstr. 1
8000 München 60

CH: 01-311 59 59
Softwareland AG
Franklinstr. 27
8050 Zürich

A: 05575-4513
Intercomp
Gschwend 163
6932 Langen b. Bregenz

Coupon

Einsenden an: Softwareland AG, Franklinstr. 27, CH-8050 Zürich, Schweiz

Bitte senden Sie mir:

○ einen Händlernachweis
○ detaillierte Informationen

DW 3/88

Name, Vorname

Strasse, Land, Ort

zzgl. DM 5.00 SFr. 5.00 ÖS 35.00 Versandkosten, unabhängig von der bestellten Stückzahl.

○ per Nachnahme ○ Verrechnungscheck liegt bei

```
    PRINT#1,Ecke(x,0)/Teiler;
    PRINT#1,Ecke(x,1)/Teiler;
    PRINT#1,Ecke(x,2)/Teiler
  NEXT x
  FOR x=0 TO Anzflach-1
    PRINT#1,"3";Fl(x,0);Fl(x,1);Fl(x,2);Fl(x,7)
    PRINT#1,"3";Fl(x,1);Fl(x,0);Fl(x,2);Fl(x,7)
  NEXT x
  CLOSE#1
  CLS
RETURN
EDGE:
  MENU 2,7,1
  PRINT#1, "OBJEKT ";D1$;" HAT";Anzkant;"KANTEN."
  PRINT#1,
  FOR x = 0 TO Anzkant-1
    IF MENU(0)=2 AND MENU(1)=7 THEN
      CLS
      MENU 2,7,0
      RETURN
    END IF
    PRINT#1,"NR.";x;TAB(9);"VON PUNKT";
    PRINT#1,Kante(x,0);"BIS";Kante(x,1);"(";
    PRINT#1,Ecke(Kante(x,0),0);Ecke(Kante(x,0),1);
    PRINT#1,Ecke(Kante(x,0),2);")-(";
    PRINT#1,Ecke(Kante(x,1),0);Ecke(Kante(x,1),1);
    PRINT#1,Ecke(Kante(x,1),2);")"
  NEXT
RETURN
WRLD:
  mode$(0)=" none"
  mode$(1)=" solid"
  mode$(2)=" checkered"
  PRINT#1, "UMGEBUNG VON OBJEKT ";D1$
  PRINT#1,
  PRINT#1, "BODEN                =";mode$(groundmode)
  PRINT#1, "HIMMEL               =";mode$(skymode)
  PRINT#1, "SCHACHBRETTMUSTER    =";Dimension;"X";Dimensi
on
  PRINT#1, "HINTERGRUNDFARBEN    = R";Rilu;"G";Gilu;"B";B
ilu
  PRINT#1, "BODENFARBE 1         = R"Rgr1;"G"Ggr1;"B";Bgr
1
  PRINT#1, "BODENFARBE 2         = R";Rgr2;"G";Ggr2;"B";B
gr2
  PRINT#1, "HIMMELFARBE 1        = R";Rsk1;"G";Gsk1;"B";B
sk1
  PRINT#1, "HIMMELFARBE 2        = R";Rsk2;"G";Gsk2;"B";B
sk2
RETURN
LAMP:
  PRINT#1, "ANZAHL LAMPEN";Anzlamp
  PRINT#1,
  FOR x=1 TO Anzlamp
    PRINT#1, "LAMPE NR.";x;"AN POSITION";Lampe(x,0);Lam
pe(x,1);Lampe(x,2)
    PRINT#1, "HELLIGKEIT";Lampe(x,3)
    PRINT#1, "FARBE     ";Lampe(x,4);Lampe(x,5);Lampe(x
,6)
  NEXT x
RETURN

VERT:
  MENU 2,7,1
  PRINT#1, "OBJEKT ";D1$;" HAT";Anzpunkte;"ECKPUNKTE."
  PRINT#1,
  FOR x = 0 TO Anzpunkte-1
    IF MENU(0)=2 AND MENU(1)=7 THEN
      CLS
      MENU 2,7,0
      RETURN
    END IF
    PRINT#1, "  ECKPUNKT NR.";x,"X =";Ecke(x,0),"Y =";E
cke(x,1),"Z ="Ecke(x,2)
  NEXT
  PRINT#1,
```

```
  PRINT#1, "MAXIMALWERT";Max
  PRINT#1, "MINIMALWERT";Min
RETURN

OBSV:
  RESTORE mode:FOR I=0 TO 3: READ mode$(I):NEXT
  RESTORE size:FOR I=0 TO 4: READ size$(I):NEXT
  RESTORE lens:FOR I=0 TO 3: READ lens$(I):NEXT
  mode:DATA painting,snapshot,photo,wireframe
  size:DATA tiny,small,medium,full,jumbo
  lens:DATA normal,wideangle,telephoto,special
  alias$(0)=" none"
  alias$(1)=" good"
  alias$(2)=" best"
  res$(0)=" lo-res"
  res$(1)=" hi-res"
  PRINT#1, "BEOBACHTER VON OBJEKT ";D1$
  PRINT#1,
  PRINT#1, "MODUS              = ";mode$(mode AND 15)
  PRINT#1, "BRENNWEITE         =";lens
  PRINT#1, "STANDPUNKT AN       ";xloc ;yloc ;zloc
  PRINT#1, "BLICKPUNKT AN       ";xtar ;ytar ;ztar
  PRINT#1, "AUFLÖSUNG          =";res$(res)
  PRINT#1, "INTERLACE          =";
  IF inter THEN
    PRINT#1, " EIN"
    ELSE
    PRINT#1, " AUS"
  END IF
  PRINT#1, "LENSMODE           = ";lens$(lmode)
  PRINT#1, "Expomode           =";
  IF exmode THEN
    PRINT#1," Manual"
    ELSE
    PRINT#1," Auto"
  END IF
  PRINT#1, "BRENNWEITE man. =";lens2
  PRINT#1, "EXPOSURE           =";expo1
  PRINT#1, "EXPOSURE  spec. =";expo2
  PRINT#1, "BILDGRÖSSE         = ";size$(size)
  PRINT#1, "KIPPWINKEL KAM= ";tilt
  PRINT#1, "ANTIALIAS          =";alias$(alias)
RETURN

FACE:
  MENU 2,7,1
  RESTORE Text
  FOR I=0 TO 4:READ Texture$(I):NEXT
  Text:DATA dull,shiny,mirror,luminous,glass
  PRINT#1, "OBJEKT ";D1$;" HAT";Anzflach;"FLÄCHEN."
  PRINT#1,
  FOR x = 0 TO Anzflach-1
    IF MENU(0)=2 AND MENU(1)=7 THEN
      CLS
      MENU 2,7,0
      RETURN
    END IF
    PRINT#1, "NR.";x;"VON PUNKT NR.";Fl(x,0);"BIS NR.";
Fl(x,1);"BIS NR.";Fl(x,2)
    PRINT#1, "FARBE =R";Fl(x,3);"G";Fl(x,4);"B";Fl(x,5)
    PRINT#1, "OBERFLÄCHE = ";Texture$(Fl(x,6) AND 15)
    PRINT#1, "WEICHZEICHNER";
    IF Fl(x,6) AND 128  THEN
    PRINT#1, " EIN"
  ELSE
    PRINT#1, " AUS"
  END IF
  PRINT#1,
  NEXT x
RETURN

Farben:
  MENU 3,5,1
  SCREEN 1,640,200,4,2
  Titel$=SPACE$(20)+"FARBEN VON OBJEKT '"+D1$+"' EDITIE
REN"
  WINDOW 3,Titel$,,0,1
```

```
   PALETTE 4,0,0,0                        'black
   PALETTE 5,0,32/255,80/255              'dark blue
   PALETTE 6,0,64/255,0                   'dark green
   PALETTE 7,80/255,0,0                   'dark red
   PALETTE 8,80/255,48/255,0              'brown
   PALETTE 9,48/255,48/255,48/255         'grey
   PALETTE 10,48/255,192/255,240/255 'light blue
   PALETTE 11,48/255,240/255,48/255  'light green
   PALETTE 12,240/255,64/255,64/255  'light red
   PALETTE 13,240/255,240/255,48/255 'yellow
   PALETTE 14,240/255,240/255,240/255'white
   PALETTE 15,.9,0,0
   LOCATE 2,28:PRINT "VIDEOSCAPE FARB-PALETTE"
   LINE (20,22)-(604,48),,b
   FOR I=4 TO 14
     LINE (21+53*(I-4),23)-(53*(I-3)+21,47),I,bf
   NEXT
   LINE (208,78+16)-(407,108+16),15,bf
   FOR x=0 TO Anzflach-1
     LOCATE 11,27:PRINT USING "_FLÄCHE NR. ####_ HAT FAR
BE";x+1
     IF x=0 THEN FARBWAHL
     IF Fl(x,3)<>Fl(x-1,3) OR Fl(x,4)<>Fl(x-1,4) OR Fl(x
,5)<>Fl(x-1,5) THEN
     FARBWAHL:
       PALETTE 15, Fl(x,3)/255,Fl(x,4)/255,Fl(x,5)/255
       LOCATE 8,22:PRINT "BITTE NEUE FARBE MIT DER MAUS
AUSWÄHLEN !"
       col=0
       WHILE col<4 OR MOUSE(0)=0 OR MOUSE(2)>47
         IF MENU(0)=3 AND MENU(1)=5 THEN
           CLS
           MENU 3,5,0
           SCREEN CLOSE 1
           RETURN
         END IF
         col = POINT(MOUSE(1),MOUSE(2))
         col=Vcol(col)+16*(Fl(x,6)AND 15)-ABS((Fl(x,6)AN
D 15)>=2)*16
       WEND
       LOCATE 8,22:PRINT "
           "
     END IF
     Fl(x,7)=col
   NEXT
   SCREEN CLOSE 1
RETURN

Ausgabe:
  LOCATE 2,9:PRINT "Sollen die Daten auf dem Drucker au
sgegeben werden (J/N) ?"
   JN$=""
   WHILE JN$<>"J" AND JN$<>"N"
     JN$=UCASE$(INKEY$)
     IF JN$="J" THEN device$="PAR:"
     IF JN$="N" THEN device$="SCRN:"
   WEND
   CLS
RETURN
Default:
   CLS
   PRINT
   PRINT " MATTE";TAB(18)"GLOSSY";TAB(34)"UNSHADED";TAB(
50)"UNFILLED OUTLINE"
   PRINT
   FOR x=0 TO 15
   PRINT x;Vcol$(x)TAB(17)x+16;
   PRINT Vcol$(x);TAB(33)x+32;
   PRINT Vcol$(x);TAB(49)x+48;Vcol$(x)
   NEXT
   LOCATE 21,2:PRINT "VOREINGESTELLT FÜR VIDEOSCAPE-FARB
E =";
   PRINT VSdef;vmode$(FIX(VSdef/16));SPC(1)Vcol$(VSdef-F
IX(VSdef/16)*16)
   LOCATE 23,2:INPUT;"ERSETZEN DURCH NR.:",col$
   IF col$<>"" THEN
```

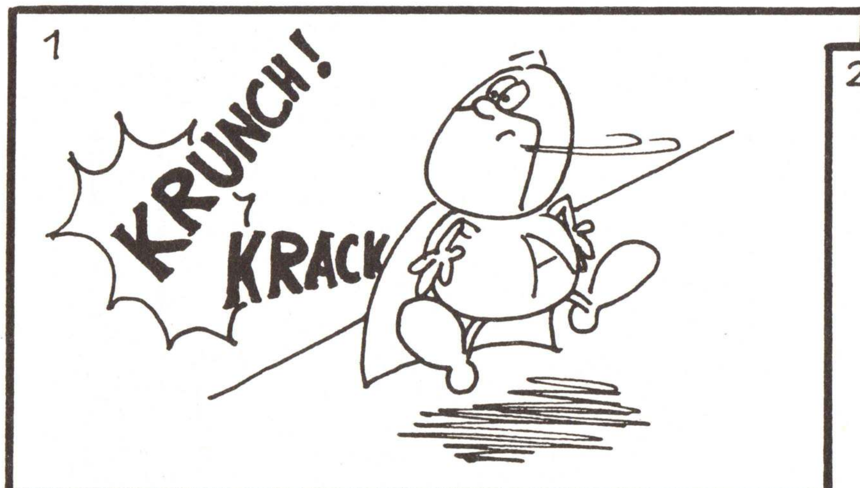
```
    col=VAL(col$)
    IF col<0 OR col>63 THEN
      CLS
      PRINT "DIESER WERT WIRD VON VIDEOSCAPE NICHT AKZE
PTIERT !"
      FOR I=0 TO 5000:NEXT
      GOTO Default
    END IF
    IF col-FIX(col/16)*16=8 THEN col=col-8 'statt code
8 wird 0 benutzt
    VSdef=col
    FOR x=0 TO Anzflach-1
    Fl(x,7)=VSdef
    NEXT
  END IF
  LINE (0,160)-(600,184),0,bf
  LOCATE 21,2:PRINT "VOREINGESTELLT FÜR VIDEOSCAPE-FARB
E =";
  PRINT VSdef;vmode$(FIX(VSdef/16));SPC(1)Vcol$(VSdef-F
IX(VSdef/16)*16)
RETURN
ACHSEN:
  CLS
  PRINT
  PRINT "WELCHE ACHSEN SOLLEN VERTAUSCHT WERDEN ?"
  INPUT "ACHSE";A1$
  INPUT "MIT";A2$
  IF A1$="" OR A2$="" THEN ACHSEN
  A1=ASC(UCASE$(A1$))-88
  A2=ASC(UCASE$(A2$))-88
  IF A1<0 OR A1>3 THEN ACHSEN
  IF A2<0 OR A2>3 THEN ACHSEN
  FOR x = 0 TO Anzpunkte-1
    SWAP Ecke(x,A1),Ecke(x,A2)
  NEXT
  CLS
RETURN
Dimension:
  CLS
  PRINT "DIMENSION"
  PRINT
  PRINT "GRÖßTER KOORDINATENWERT   =";Max
  PRINT "KLEINSTER KOORDINATENWERT =";Min
  PRINT "VORGESCHLAGENER TEILER    =";Teiler
  PRINT
  PRINT "ERGIBT FÜR GRÖßTEN KOORDINATENWERT   =";Max/Te
iler
  PRINT "ERGIBT FÜR KLEINSTEN KOORDINATENWERT =";Min/Te
iler
  PRINT
  PRINT "Wollen Sie den Teiler modifizieren (J/N) ?"
  a$=""
  WHILE a$<>"J" AND a$<>"N"
  a$=UCASE$(INKEY$)
  WEND
  IF a$="J" THEN
    LOCATE 10,1:INPUT "Bitte neuen Teiler eingeben
    ";NTeiler
    IF NTeiler=0 THEN NTeiler=Teiler
    PRINT
    PRINT "ERGIBT FÜR GRÖßTEN KOORDINATENWERT   =";Max/
NTeiler
    PRINT "ERGIBT FÜR KLEINSTEN KOORDINATENWERT =";Min/
NTeiler
    Teiler=NTeiler
  END IF
RETURN
Anzeige:
  CLS
  MENU 3,5,1
  PRINT "Anzeige"
  w=1/SQR(2)
  FOR I = 0 TO Anzflach-1
    IF MENU(0)=3 AND MENU(1)=5 THEN
      CLS
      MENU 3,5,0
      RETURN
    END IF
```

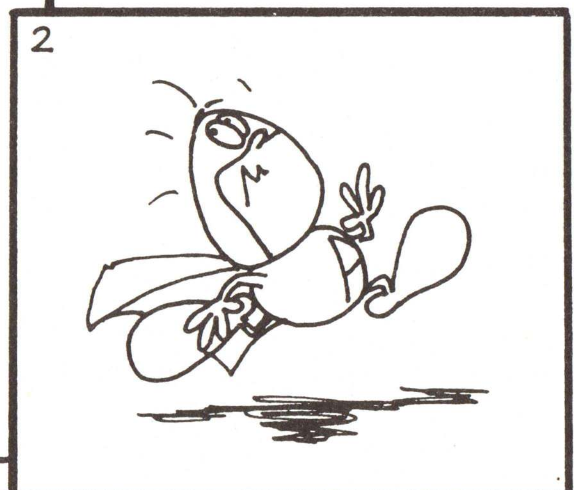
```
      FOR j=0 TO 2
        x = (Ecke(Fl(I,j),0)/Teiler)
        y = w*(Ecke(Fl(I,j),1)/Teiler)-w*(Ecke(Fl(I,j),2)
/Teiler)
        z = w*(Ecke(Fl(I,j),1)/Teiler)+w*(Ecke(Fl(I,j),2)
/Teiler)
        von(j) = 320+(-(w*x-w*z)/((y-250)/200))*2
        bis(j) = 100+(-(w*x+w*z)/((y-250)/200))
      NEXT j
      LINE(von(0),bis(0))-(von(1),bis(1))
      LINE -(von(2),bis(2))
      LINE -(von(0),bis(0))
    NEXT I
RETURN
```

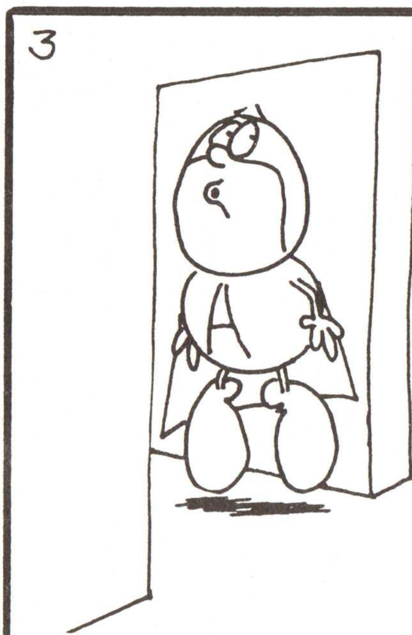
AMIGA MAN

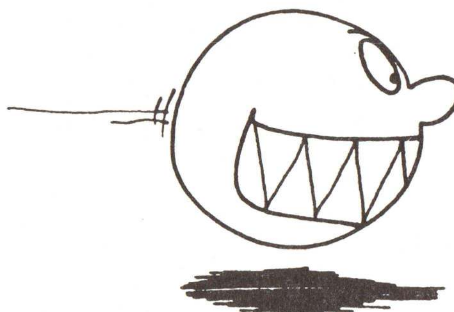
ABENTEUER IM AMIGA!













VON ANDREAS DIEBOLD

BILDER DIFFERENZIERT

Ein Verfahren zur Komprimierung von IFF-Bild-Animationen

Dieses Programm ermöglicht ein schnelles aufeinanderfolgendes Zeigen von IFF-Bildern, die zusammen eine Animation ergeben. Es werden nicht einfach komplette Bildschirminhalte herumkopiert, sondern die Differenzen zwischen den einzelnen Bildern erfaßt. Eine Animation sollte ein fließender Ablauf sein, d.h., daß sich von einem Bild zum nächsten die Lage eines Objektes auf dem Bildschirm nicht wesentlich ändert. Eine deutliche Bewegung erfolgt nur über viele kleine Bewegungen im Laufe vieler Bilder. Normalerweise bietet der Arbeitsspeicher dafür aber nicht genug Platz, da allein eine Grafik schon mehrere 10000 Byte einnehmen kann. Ist nun aber die Veränderung von einem Bild zum nächsten nicht sehr groß, so ist es vernünftiger, nur den Unterschied zwischen den Bildern festzuhalten, indem man beide in den Speicher holt und überprüft, inwieweit sich die Bytefolgen unterscheiden.

Genau das passiert auch in diesem Programm. Es lädt ein Startbild und öffnet dementsprechend einen Screen. Danach wird dieses Bild in einen zweiten Speicherbereich kopiert, der dem Speicherbereich eines Screens gleich ist. Nun kann das nächste Bild in den Screen geladen werden, während das vorhergehende ja in einem zweiten Bereich weilt. Nachdem das nächste Bild geladen wurde, werden alle WORDS (16 Bit-Worte) beider Speicherbereiche miteinander verglichen. Dies geschieht für jede BitPlane eines Bildes. In einem weiteren Speicherbereich werden nun diese Differenzdaten abgelegt, und zwar zuerst die Anzahl der Änderungen der 1. Bitplane zwischen Bild eins und zwei. Danach folgen diese Änderungen, jede einzelne in Form von zwei WORDS. Das erste WORD gibt an, welches WORD in der BitPlane sich zwischen den Bildern unterscheidet. Das Zweite gibt den neuen Inhalt für diese Position an. Danach geht es mit den nächsten Planes weiter, bis diese Bilder fertig sind. Dann geht es für noch mehr Bilder genauso weiter. Das zweite Bild wird in den Hilfsspeicher und ein Drittes in den Screen geladen. Natürlich verschwinden die Bilder dadurch aus dem Speicher, aber in den Differenzdaten bleiben sie erhalten. Wichtig dabei ist, daß alle Bilder dieselbe Auflösung und Farbtiefe haben. Die Farben werden immer

vom ersten Bild übernommen. So könnten die Differenzdaten aussehen, für Bilder mit zwei Bitplanes (alle Differenzdaten sind 16 Bit lang):

1. *00002 (= 2 Änderungen in der 1.Plane)*
2. *00623 (= Das 623. WORD hat sich geändert)*
3. *65321 (= Neuer Inhalt des 623. Wortes)*
4. *00624 (= Das 624. Word ...)*
5. *01234 (= Neuer Inhalt)*
6. *00000 (= 0 Änderungen in der 2.Plane)*

Nun beginnt ein neues Bild, da die Zahl der Bitplanes feststeht.

7. . . .

n. *65535 = Ein Abschlußwort (HEX FFFF), welches das Ende markiert).*

Um nun diese Bildfolge zu zeigen, muß nur das Startbild geladen werden. Danach werden die Differenzdaten gelesen und dementsprechend die Speicherinhalte des Bildschirmspeichers geändert, also z.B. auf das 623. Word der 1.Plane der Wert 65321 geschrieben.

Bei nur geringen Änderungen wird der benötigte Platz für ein Bild sehr gering. Außerdem erhöht sich in diesem Falle die Bildwiedergabegeschwindigkeit enorm. Das Programm ermöglicht weiterhin, die Differenzdaten zu speichern. Diese werden dann unter einem einzugebenden Namen zusammen mit dem Namen des dazugehörigen Startbildes gespeichert. Werden sie dann wieder geladen, benötigt man auch den File des ersten Bildes. Bei einer größeren Anzahl von Änderungen kann es vorkommen, daß das Bild flackert, während die Änderungen in den Bildschirmspeicher geschrieben werden. Um dies zu beseitigen, gibt es den Dbuf-Mode, der aber einenzweiten Screen benötigt und daher mehr Speicher kostet.

Das Programm ist weniger dafür gedacht, vollkommen unterschiedliche Bilder hintereinander zu zeigen, da die Änderungsdaten dann einen riesigen Umfang annehmen würden. Bei geringen Änderungen aber, wie z.B. in den hier abgebildeten Bildern, erreichensie sehr hohe Ablaufgeschwindigkeiten bei geringem Speicherverbrauch.

Durch das Programm führt ein einfaches Menü. Nach dem Tastendruck <f> nennt man den Namen eines Textfiles, der in der vorgesehenen Reihenfolge alle Bilder enthält, welche zu dem Ablauf gehören. Um ein Wiederholen zu ermöglichen, sollte das letzte Bild dasselbe wie das erste sein. Als Abschluß wird einfach <#> eingegeben. Diesen File kann man auch mit einem normalen Editor erzeugen. Danach wird mit <m> die Animation erzeugt. Man gibt entweder den Pfadnamen zu einem oben beschriebenen Textfile an, oder man aktiviert den zuletzt bei <f> Edierten. Während der Erstellung sieht man alle Bilder. Sollte ein Bild nicht unter dem im Textfile angegebenen Namen zu finden sein, stoppt das Programm, und man hat die Chance, diesen Namen zu korrigieren. Ist auch dieser Teil abgeschlossen und man in das Menü zurückgekehrt, so können die Animationsdaten mit <s> gespeichert werden. Einen gespeicherten Animationsdatensatz kann man mit <l> laden. Um nun die Animation zu zeigen, muß sichergestellt sein, daß das Startbild in der aktuellen Directory verfügbar ist. Danach drückt man <a>, und nach kurzer Zeit kann man die Anzahl der Wiederholungen und der Pausen zwischen den Bildern festlegen. Dann werden die Bilder gezeigt, so schnell es eben möglich ist. Aus dem Wiedergabemodus kommt man durch Eingabe eines negativen Wertes für die Bildpausen oder Wiederholungen heraus und in das Menü zurück. Der Dbuf-Mode, der ein Flackern des Bildes bei großen Änderungen vermeidet, wird mit <d> ein und ausgeschaltet. Zum Beenden des Programms dient die Taste <e>. Kompiliert wurde das Programm mit Aztec-C 3.4a und folgenden Compiler- bzw. Linkeraufrufen:

CC picsw.c und LN picsw.o -lc.

```
==================================================================
LISTING  VON :                                        Seiten ____
picswitch.c                    21159 rwed 16-Dec-87 18:53:10
=========================================================== 3.4 a ======

#include <exec/types.h>
#include <intuition/intuitionbase.h>
#include <functions.h>
#include <exec/memory.h>
#include <libraries/dos.h>

 /* defines in eine Zeile schreiben */
#define POS ((( zeile * scr_data->pic_width )
                      / 8 ) + ( xp / 8 ))
#define BMHD (STRPTR)"BMHD"
#define CMAP (STRPTR)"CMAP"
#define BODY (STRPTR)"BODY"
#define CAMG (STRPTR)"CAMG"
#define PLANELOOP for(i=0;i<firstpic->pic_depth
                                          ;i++)
#define DISP_PLANE scr->BitMap.Planes[i]
#define BUFF_PLANE planebuf[i]
#define GRAF_BASE (struct GfxBase *)
        OpenLibrary("graphics.library",0L);
#define INTU_BASE (struct IntuitionBase *)
        OpenLibrary("intuition.library",0l);

struct iffpic
 {
   ULONG form_len;
   ULONG bmhd_len;
   ULONG cmap_len;
   ULONG body_len;
   UBYTE *form;
   UBYTE *bmhd;
   UBYTE *cmap;
   UBYTE *body;
   UWORD pic_width;
   UWORD pic_height;
   WORD pic_xoff;
   WORD pic_yoff;
   UBYTE pic_depth;
   UBYTE pic_mask;
   UBYTE pic_comp;
   UBYTE unused;
   UWORD pic_transp_col;
   UBYTE xAsp;
   UBYTE yAsp;
   UWORD screenX;
   UWORD screenY;
 };

ULONG *vmp,viewmodes,test,anim_len,planesize,
      displaysize,anlen;

UWORD colortable[32],*anpos,*animdata,dbuf = 0,
      colnum;
UBYTE header[12],bmh[28],file[70],firstfile[70],
      namebuf[70],*suche_chunk(),tempfile[70],
      getkey();
PLANEPTR planebuf[6];
struct FileHandle *fh,*iff_file,*open_file();

struct iffpic *firstpic,*nextpic;
struct BitMap customBitMap;
struct Screen *scr,*bscr,*open_scr();
struct NewScreen nscr;
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Window *dw;

main()
 {
  UBYTE c;

  startup();

  FOREVER
   {
    printf("\n\nBENUTZUNGSHINWEIS :\n");
    printf(" (m)ache eine Animation\n");
    printf(" (a)nimiere\n");
    printf(" (f)ile erstellen\n");
    printf(" (l)aden einer Anim.\n");
    printf(" (s)peichern einer Anim\n");
    printf(" (e)nde des Programms\n");
    printf(" (d) DBUF ein/aus momentan :");
    if(dbuf) printf(" EIN\n");
    else     printf(" AUS\n");
    printf("\n TMP : %s\n",tempfile);

    switch(getkey(dw))
     {
      case 'd':
               dbuf = dbmode();
               break;
      case 'm':
               make_anim();
               break;
      case 'a':
               show_anim();
               break;
      case 'f':
               fastfile();
               break;
      case 'l':
               if(load_anim())
                printf("laden mi■lungen\n");
               else
                printf("laden gelungen\n");
               break;
      case 's':
               if(save_anim())
                printf("speichern mi■lungen\n");
               else
                printf("speichern gelungen\n");
               break;
      case 'e':
               ende ("Programm Ende");
               break;
      default:
              break;
     }
   }
 }

/**********************************************
 *
 *     CHUNK in den IFF Daten suchen
 *
 **********************************************/

UBYTE *suche_chunk(chunk_name,lpic)

  STRPTR chunk_name;
  struct iffpic *lpic;
  {
   UBYTE *such_ptr;

   for(such_ptr = lpic->form
       ;such_ptr < (lpic->form + lpic->form_len);
        such_ptr++)
    {
     if( *such_ptr      == *chunk_name &&
         *(such_ptr+1) == *(chunk_name+1) &&
         *(such_ptr+2) == *(chunk_name+2) &&
         *(such_ptr+3) == *(chunk_name+3)    )
       return(such_ptr);
    }
   return(NULL);
  }


/**********************************************
 *
 *   Globaler Ausstieg
 *
 **********************************************/
ende(why)
 char *why;

  {
   SHORT i;
   printf("%s",why);

   if(fh)      Close(fh);
   if(iff_file)Close(iff_file);
   if(firstpic)FreeMem(firstpic
                 ,(LONG)sizeof(struct iffpic));
   if(nextpic)FreeMem(nextpic
                 ,(LONG)sizeof(struct iffpic));
   if(animdata)FreeMem(animdata,anim_len);
   if(scr)CloseScreen(scr);
   if(bscr)CloseScreen(bscr);
   if(GfxBase)       CloseLibrary(GfxBase);
   if(IntuitionBase) CloseLibrary(IntuitionBase);
   printf("\n\n ENDE \n\n");
   Exit(1L);
  }


/**********************************************
 *
 *   Screen +ffnen
 *
 **********************************************/

struct Screen *open_scr()
 {
  struct Screen *oscr;

  planesize = RASSIZE((LONG)firstpic->pic_width,
                      (LONG)firstpic->pic_height);

  nscr.LeftEdge      = firstpic->pic_xoff;
  nscr.TopEdge       = firstpic->pic_yoff;
  nscr.Width         = firstpic->pic_width;
  nscr.Height        = firstpic->pic_height;
  nscr.Depth         = firstpic->pic_depth;
  nscr.DetailPen     = NULL;
  nscr.BlockPen      = NULL;

  if(firstpic->pic_width > 320)
   {
    if(firstpic->pic_height > 256)
     nscr.ViewModes = HIRES|LACE;
    else
     nscr.ViewModes = HIRES;
   }
  else
   {
    if(firstpic->pic_depth == 6)
     {
      if(firstpic->pic_height > 256)
       nscr.ViewModes = HAM|LACE;
      else
       nscr.ViewModes = HAM;
     }
    if(firstpic->pic_height > 256)
     nscr.ViewModes = LACE;
    else
     nscr.ViewModes     = NULL;
   }
  if(viewmodes)nscr.ViewModes = viewmodes;

  nscr.Type          = CUSTOMSCREEN;
  nscr.Font          = NULL;
  nscr.DefaultTitle  = NULL;
```

```
 nscr.Gadgets        = NULL;
 nscr.CustomBitMap   = NULL;

 if(!(oscr = (struct Screen *)OpenScreen(&nscr)))
     ende("kein Screen");


 return(oscr);
}


/**********************************************
 *
 *  Dekomprimieren und in Bildschirmspeicher
 *
 **********************************************/

read_body(scr_data,disp)
 struct iffpic *scr_data;
 struct Screen *disp;
 {
 LONG n,zeile;
 struct BitMap *bm;
 BYTE *bp,m,plane;
 WORD spalte,xp;

 bm = (struct BitMap *)&(disp->BitMap);
 bp = (BYTE *)scr_data->body ;
 bp += 8;

 for(zeile = 0;zeile < scr_data->pic_height
                                    ;zeile++)
  {
   for(plane = 0;plane < scr_data->pic_depth
                                    ;plane++)
    {
     spalte = NULL;
     while(spalte <  (scr_data->pic_width - 8) )
      {
       m = *bp;
       if(m != -128)
        {
         if(m < 0)
          {
           m *= -1;
           bp++;
           for(xp = spalte
               ;xp <=  (spalte + (m*8));xp += 8)
            {
             *((bm->Planes[plane]) + POS) = *bp;
            }
           spalte = xp;
            bp++;
          }
         else
          {
           for(xp = spalte
               ;xp <=  (spalte + (m*8));xp += 8)
            {
             *((bm->Planes[plane]) + POS)
                                = *(++bp);
            }
           spalte = xp;
           bp++;
          }

        }  /* END of if(-128       */
       else bp++;
      }     /* END of while(spalte */

    }       /* END of for(zeile    */

  }         /* END of for(plane    */

  Delay (50L);
 FreeMem(scr_data->form,scr_data->form_len);
 scr_data = NULL;
 }          /* END of function     */

/**********************************************
 *
 *  FARBEN AUF SCREEN SETZEN
 *
 **********************************************/

set_colors(screen)
 struct Screen *screen;
 {
 UBYTE *colval;
 UWORD color,i;
 LONG line = 0;

 colnum = 2;
 if(firstpic->pic_depth < 6)
  {
   PLANELOOP
      colnum*=2;
  }
 else colnum = 16;

 colval = ((UBYTE *)firstpic->cmap)+8;

 for(i=0;i<colnum;i++)
  {
   color = 0;
   color += (*colval++)*16;
   color += (*colval++);
   color += (*colval++)/16;

   colortable[i] = color;
  }

 LoadRGB4(&scr->ViewPort,colortable,
                          (LONG)colnum);
 return();
}



/**********************************************
 *
 *  LESEN + INITIALISIERUNG DER IFF STRUCT
 *
 **********************************************/

read_IFF(rpic,pfn)
 struct iffpic *rpic;
 STRPTR pfn;
 {
 LONG act_size;

 iff_file = open_file(pfn);
 if(!iff_file)return();

 Read(iff_file,&header[0],12L);

 if( *((ULONG *)&header[0]) != 0x464f524dL)
     ende("FORM ERR");

 if( *((ULONG *)&header[8]) != 0x494c424d)
     ende("ILBM_ERR");

 rpic->form_len = *((ULONG *)&header[4]);

 rpic->form = (UBYTE *)AllocMem
                  (rpic->form_len,MEMF_CLEAR);
 if(!rpic->form)ende("GRAFIK_ERR");
 act_size = Read(iff_file ,rpic->form
                          ,rpic->form_len);

 if((rpic->bmhd = suche_chunk(BMHD,rpic))==NULL)
     ende("BMHD ERR");
 if((rpic->cmap = suche_chunk(CMAP,rpic))==NULL)
     ende("CMAP ERR");
 if((rpic->body = suche_chunk(BODY,rpic))==NULL)
     ende("BODY ERR");
 if(vmp = (ULONG *)suche_chunk(CAMG,rpic ))
   viewmodes = *(vmp+2);
 else viewmodes = NULL;

 rpic->bmhd_len  =*((ULONG *)((rpic->bmhd)+4));
 rpic->cmap_len  =*((ULONG *)((rpic->cmap)+4));
 rpic->body_len  =*((ULONG *)((rpic->body)+4));
 rpic->pic_width =*((UWORD *)((rpic->bmhd)+8));
 rpic->pic_height=*((UWORD *)((rpic->bmhd)+10));
 rpic->pic_xoff  =*( (WORD *)((rpic->bmhd)+12));
 rpic->pic_yoff  =*( (WORD *)((rpic->bmhd)+14));
 rpic->pic_depth =*((UBYTE *)((rpic->bmhd)+16));
 rpic->pic_mask  =*((UBYTE *)((rpic->bmhd)+17));
 rpic->pic_comp  =*((UBYTE *)((rpic->bmhd)+18));
 rpic->xAsp      =*((UBYTE *)((rpic->bmhd)+22));
 rpic->yAsp      =*((UBYTE *)((rpic->bmhd)+23));
 rpic->screenX   =*((UWORD *)((rpic->bmhd)+24));
 rpic->screenY   =*((UWORD *)((rpic->bmhd)+26));

 return();
 }


/**********************************************
 *
 *  Oeffnen eines Buffers fuer den Screen
 *
 **********************************************/

get_planebuffer()
 {
 WORD i;

 PLANELOOP
  {
   planebuf[i] = (PLANEPTR)AllocMem
                          (planesize,NULL);
   if(!planebuf[i])ende("no Buffer Planes" );
  }
 }


/**********************************************
 *
 *  Kopieren des Screens in den Buffer
 *
 **********************************************/

copy_screen()
 {
 WORD i;

 PLANELOOP
   CopyMemQuick(DISP_PLANE,BUFF_PLANE,planesize);
 return();
 }

/**********************************************
 *
 *  Vergleichen der Screen Dimensionen
 *
 **********************************************/

cmp_iffdata(dat1,dat2)
 struct iffpic *dat1,*dat2;
 {
  if( dat1->pic_width  == dat2->pic_width  &&
      dat1->pic_height == dat2->pic_height &&
      dat1->pic_depth  == dat2->pic_depth   )

     return(TRUE);
```

```
  else return(FALSE);
 }


/***********************************************
 *
 *   Erstellung der Animationsdaten
 *
 ***********************************************/

check_diff()
 {
  UWORD  i,pos,*countpos;

  PLANELOOP
   {
    countpos = anpos++;
    if(anpos > (animdata+anim_len-2))return(1);
    *countpos = 0;

    for(pos = 0;pos < (planesize);pos+=2)
     {
      if(*((UWORD *)(DISP_PLANE+pos)) !=
          *((UWORD *)(BUFF_PLANE+pos)))
       {
        (*countpos)++;

        if(anpos > (animdata+anim_len-2))
                                        return(1);
        *anpos++ = pos;
        *anpos++ = *((UWORD *)(DISP_PLANE+pos));
       }
     }  /* end of for (pos  */
   }    /* end of PLANELOOP */
  return(0);
 }


/***********************************************
 *
 *   Wiedergaberoutine der Animation
 *
 ***********************************************/

playanim(pause,wieoft)
 LONG pause;
 LONG wieoft;
 {
  UWORD *anim,*pos,i,change,chnum;
  LONG sooft;

  if(dbuf)ScreenToFront(bscr);

  for(sooft=0;sooft<wieoft;sooft++)
   {
    anim = animdata;
    while(*anim != 0xFFFF)
     {
      PLANELOOP
       {
        chnum = *anim++;

        for(change = 0;change < chnum;change++)
         {
          pos = (UWORD *)(DISP_PLANE+(*(anim++)));
          *pos = *(anim++);
         }
       }
      Delay(pause);
      if(dbuf) copy_screen();
     }
   }
  return(NULL);
 }

/***********************************************
 *
 *   Kontolliertes oeffnen eines Files
 *
 ***********************************************/


struct FileHandle * open_file(filename)
 STRPTR filename;
 {
  struct FileHandle *filehandle;
  struct Lock *lock;

  FOREVER
   {
    if(lock = Lock(filename,ACCESS_READ))
     {
      UnLock(lock);
      filehandle = Open(filename,MODE_OLDFILE);
      if(filehandle)return(filehandle);
     }
    printf("File: %s nicht vorhanden !!!\n"
              ,filename);
    printf("NOCHMAL ? ( #+ = Abbruch ) :");
    scanf("%s",filename);
    if(filename[0] == '#' && filename[1] == '+')
       return(NULL);
   }
 }

/***********************************************
 *
 *   Lesen aus dem Textfile
 *
 ***********************************************/

readname(filehandle,buffer)
 struct FileHandle *filehandle;
 STRPTR buffer;
 {
  FOREVER
   {
    if( NULL == Read(filehandle,buffer,1L))
       break;
    if(*buffer == 10)
       break;
    buffer++;
   }
  *buffer = 0;
  return();
 }

/***********************************************
 *
 *   Verwaltung der Animationserstellung
 *
 ***********************************************/

make_anim()
 {
  WORD i;

  if(dbuf)dbuf = FALSE;
  if(scr)CloseScreen(scr);
  if(bscr)CloseScreen(bscr);
  scr = bscr = NULL;

  printf("\n\nWelcher Textfile  ");
  printf("( #t = letzter Tempfile ?:\n");
  scanf("%s",namebuf);
  if( namebuf[0] == '#' && namebuf[1] == 't' )
   strcpy(namebuf,tempfile);

  fh = open_file(namebuf);
  if(!fh)return();

  readname(fh,namebuf);

  /* den Filenamen des 1.Bildes festhalten */
  strcpy(firstfile,namebuf);

  /* nun alle IFF Daten in den Speicher holen */
  read_IFF(firstpic,firstfile);

  /* entsprechend den Daten des 1.Bildes die Daten
   * nun einen Screen aufmachen, die Farben setzen
   * und einen Screen-Buffer bereitstellen */
  scr = open_scr();
  set_colors(scr);

  /* nun die IFF-Bilddaten lesen und das Bild auf
   * den Screen bringen */
  read_body(firstpic,scr);

  /* und das ganze im Screen-Buffer sichern */
  get_planebuffer();
  copy_screen();

  /* Das Start Bild ist nun geladen, nun folgt
   * der Rest */

  FOREVER
   {
    WBenchToFront();

    /* Name des folgenden Bildes aus dem Textfile
       holen */
    readname(fh,namebuf);

    /* Testen ob Ende erreicht */
    if(namebuf[0] == '#')break;

    /* Das Laden der folgenden Bilder erfolgt so
       wie beim 1.Bild, nur wird kein Screen
       geoeffnet und die Farben bleiben immer
       die des ersten Bildes... */
    read_IFF(nextpic,namebuf);

    /* ... nur wird getestet ob das neue Bild die-
          selben Dimensionen hat */
    if(cmp_iffdata(firstpic,nextpic))
     {
      ScreenToFront(scr); /* damit man es sieht */
      read_body(nextpic,scr);

      /* Nun erfolgt die Differenzenerstellung
         zum vorhergehenden Bild. Ist nicht
         genug Platz fuer diese Differenzenequenz
         so wird dies erkannt und abgebrochen  */
      if(check_diff())

       /* Wenn zu wenig Speicher */
       ende("Out of Memory");

      copy_screen();
     }
    else
     {
      printf("Dieses Bild entsprach nicht dem");
      printf("Startbild und wird ignoriert\n");
     }
   }
  WBenchToFront();


  /* An die letzte Stelle der Differenzdaten wird
     ein Abschluss WORD gesetzt */
  *anpos = 0xFFFF;

  anlen = anpos-animdata;anlen *= 2;anlen+=2;
  printf("Differenz-Daten = %ld Bytes\n",anlen);
```

```
  anpos = animdata;

  PLANELOOP
    if(BUFF_PLANE)FreeMem (BUFF_PLANE,planesize);

  if(fh) Close(fh);
  fh = NULL;

  return();
 }


/************************************************
 *
 *   Verwaltung der Animationswiedergabe
 *
 ************************************************/

show_anim()
 {
  LONG delay,loops;
  printf("Laden des Startbildes ...\n\n");
  /* erstes Bild wieder laden */
  read_IFF(nextpic,firstfile);
  read_body(nextpic,scr);


  FOREVER
   {
    WBenchToFront();
    printf("Wieviel 1/50 sek. Pause zwischen ");
    printf("den Bildern ? (Negativ = Abbruch)");
    scanf("%ld",&delay);
    if( delay < NULL )break;

    printf("Anzahl der Wiederholungen ?\n");
    printf("!! Wiederholungen sind nur dann");
    printf(" sinnvoll wenn das erste Bild ");
    printf("gleich dem letzten ist\n");
    printf("(Negativ = Abbruch)");
    scanf("%ld",&loops);
    if( loops < NULL )break;

    ScreenToFront(scr);
    playanim(delay,loops);
   }
  return();
 }

/************************************************
 *
 *   Allgemeines Oeffnen von ...
 *
 ************************************************/

startup()
 {

  /* wichtigen Libraries */
  IntuitionBase = INTU_BASE
  if(!IntuitionBase)
     ende("keine IntuitionBase");

  GfxBase = GRAF_BASE
  if(!GfxBase)
     ende("keine GfxBase");

  dw = IntuitionBase->ActiveWindow;

  /* Speicherplatz fuer IFF Daten */
  firstpic = (struct iffpic *)AllocMem
       ((LONG)sizeof(struct iffpic),MEMF_CLEAR);
  nextpic = (struct iffpic *)AllocMem
       ((LONG)sizeof(struct iffpic),MEMF_CLEAR);

  /* Speicherplatz fuer die Differenzdaten
     zuerst schauen wie gross der groesste freie
     Block ist */
  anim_len = (ULONG)
           AvailMem(MEMF_LARGEST);

  /* 60 KByte fuer den Rest uebrig lassen */
  anim_len -= 60000L;
  anpos = animdata = (UWORD *)AllocMem
                   (anim_len,MEMF_CLEAR);

  return();
 }


/************************************************
 *
 *   Textfile Erstellung
 *
 ************************************************/

fastfile()
 {
  struct FileHandle *fih;
  UBYTE co=0,LF=10,str[70];

  printf("\nName des Tempfiles ?:");
  scanf("%s",tempfile);
  fih = Open(tempfile ,MODE_NEWFILE);
  if(!fih)
   {
    printf("Kein File m+glich\n");
    return(0);
   }

  FOREVER
   {
    printf("Zeile eingeben :");
    scanf("%s",str);

    if(Write(fih,str,(LONG)strlen(str)) !=
            (LONG)strlen(str))
     { remove(fih,tempfile); return(); }

    if(Write(fih,&LF,1L) != 1L)
     { remove(fih,tempfile); return(); }

    if( str[0] == '#' )break;
   }

  if(fih)  Close(fih);
  fih = NULL;
  return();
 }


/************************************************
 *
 *   Speichern einer Animation
 *
 ************************************************/

save_anim()
 {
  struct FileHandle *fih;
  UBYTE LF=10, savename[70];

  printf("SPEICHERN ! Filename ?:");
  scanf("%s",savename);

  fih = Open(savename,MODE_NEWFILE);
  if(!fih)return(1);

  printf("schreibe name\n");
  if(Write(fih,firstfile,(LONG)strlen(firstfile))
      != (LONG)strlen(firstfile))
     { remove(fih,savename); return(1); }

  printf("schreibe linefeed\n");
  if(Write(fih,&LF,1L) != 1L)
     { remove(fih,savename); return(1); }

  printf("schreibe daten von %ld Bytes\n",anlen);
  if(Write(fih,(UBYTE *)animdata,anlen) != anlen)
     { remove(fih,savename); return(1); }

  if(fih) Close(fih);
  fih = NULL;
  return(0);
 }

/************************************************
 *
 *   Animationen Laden
 *
 ************************************************/

load_anim()
 {
  UBYTE loadname[70];
  struct FileHandle *fih;

  if(dbuf)dbuf = FALSE;
  if(scr)CloseScreen(scr);
  if(bscr)CloseScreen(bscr);
  scr = bscr = NULL;

  printf("LADEN ! Filename ?:");
  scanf("%s",loadname);

  fih = open_file(loadname);
  if(!fih)return(1);

  readname(fih,firstfile);
  printf("Name des Startbildes : %s\n",firstfile);
  read_IFF(firstpic,firstfile);
  scr = open_scr();
  ScreenToBack(scr);
  set_colors(scr);

  anlen = Read(fih,(UBYTE *)animdata,anim_len);

  return(0);
 }


/************************************************
 *
 *   Schreibgestoerte Files entfernen
 *
 ************************************************/

remove(filha,remfile)
 struct FileHandle *filha;
 STRPTR remfile;
 {
  printf("Fehler beim Schreiben in %s\n",remfile);

  Close(filha);
  filha = NULL;
  printf("gescl. fh = %ld\n",filha);

  if(DeleteFile(remfile))
    printf("File ist entfernt\n");
  else
    printf("File konnte nicht entfernt werden\n");

  return();
 }


/************************************************
```

```
 *
 *  ein gut funftionierendes getchar();
 *
 ************************************************/

UBYTE getkey(readwindow)
 struct Window *readwindow;
 {
  STRPTR cp;
  struct MsgPort *wpBuf,*upBuf;
  struct IntuiMessage *mess;

  wpBuf = readwindow->WindowPort;
  upBuf = readwindow->UserPort;
  ModifyIDCMP(readwindow,VANILLAKEY);

  FOREVER
   {
    Wait(1L<<readwindow->UserPort->mp_SigBit);

    while(mess = (struct IntuiMessage *)
                  GetMsg(readwindow->UserPort))
     {
      if(mess->Class == VANILLAKEY);
       {
        cp = (STRPTR)&mess->Code;
        ReplyMsg(mess);
        ModifyIDCMP(readwindow,NULL);
        readwindow->UserPort = upBuf;
        readwindow->WindowPort = wpBuf;
        return( *(cp+1) );
       }
      ReplyMsg(mess);
     }
   }
 }


dbmode()
 {
  WORD i;

  if(!scr)return(NULL);

  if(!dbuf)
   {
    bscr = open_scr();
    if(!bscr)
     {
      printf("\nNicht genug CHIP-RAM fuer ");
      printf(" zweiten Screen\n");
      return(0);
     }
     ScreenToBack(bscr);
    LoadRGB4(&bscr->ViewPort,colortable,
                         (LONG)colnum);
    PLANELOOP
     BUFF_PLANE = bscr->BitMap.Planes[i];
    return(1);
   }
  if(bscr)CloseScreen(bscr);
  bscr = NULL;
  return(0);
 }


                  --- END OF FILE : picswitch.c ---
=================================================================
```

VON CHRISTIAN SCHORMANN

FÜR LITERATEN

BÜCHER ZUR COMPUTERGRAFIK

Im folgenden wollen wir Ihnen eine Reihe von Einführungen, Lehrbüchern und Aufsatzsammlungen vorstellen, in denen Sie Materialien zur Computergrafik finden. Doch Vorsicht ! Wenn Sie sich näher mit dieser Materie beschäftigen wollen, sollten Sie gute Grundkenntnisse der Mathematik mitbringen. Ohne geht es gar nicht, und je komplizierter die Verfahren werden, desto mehr höhere Mathematik wird von Ihnen verlangt. Beachten Sie bitte außerdem, daß es hier nur um Bücher geht, die sich allgemein mit dem Thema beschäftigen, nicht etwa speziell auf den Amiga bezogen.

Wer eine sehr allgemeine und universelle Einführung in die Thematik sucht, sollte sich unbedingt die "Grundzüge der interaktiven Computergrafik" von William M. Newman und Robert F. Sproull, erschienen im McGraw-Hill-Verlag, ansehen. Für 54.- DM erhält man fast 600 Seiten Buch in Paperback-Form. Hier lassen sich die Autoren über so ziemlich jeden Bereich der Computergrafik aus. Die deutsche Übersetzung ist relativ neu (1986), allerdings stammt die englische Originalausgabe vom Beginn der 80er Jahre, ist also nicht immer auf dem allerneuesten Stand. Das ist für eine grundsätzliche Einführung aber auch nicht wichtig.

Das Buch enthält zahlreiche Programmbeispiele, die in PASCAL geschrieben wurden. Eine Übertragung in andere Sprachen, die Prozedur-Aufrufe zulassen, sollte kein Problem sein; die Kommentierung ist gut.

Wegen des enormen Stoffumfangs, mit dem sich das Buch beschäftigt, kommen Details, besonders bei komplizierteren Problemen, etwas zu kurz. Als Einführung in allgemeine Problemkreise ist es aber hervorragend geeignet und kann sehr empfohlen werden.

Bei Addison-Wesley gibt es ein Konkurrenzwerk mit einem sehr ähnlichen Titel, das aber leider nicht in Übersetzung vorliegt und ziemlich genau doppelt so teuer ist. Es ist etwas genauer, wenn es um die komplexeren Probleme geht, und scheint ebenfalls sehr empfehlenswert, lag uns aber nicht für eine genauere Revision vor.

Die Lücken in der oben erwähnten Einführung füllen vier andere Bücher aus dem gleichen Verlag vollständig aus.

Von David F. Rogers stammen zwei Bände im Taschenbuchformat, deren erster, "Mathematical Principles for Computer Graphics", versucht, die verschiedenen mathematischen Grundlagen der Grafikprogrammierung in einer einheitlichen Darstellung zu erklären. Gute Englisch-Kenntnisse sind unbedingt erforderlich, auch der Stil ist für totale Laien nicht gerade besonders motivierend.

Dennoch, der Inhalt des Buches ist ausführlich und gut gegliedert. Die Erklärungen sind gut, wenn man sich mit dem trockenen Stil anfreunden kann. Abgerundet wird das Ganze durch eine große Anzahl von Beispielsroutinen, die in Basic geschrieben sind, sowie einen kurzen Anhang, in dem grundsätzliche Prinzipien der Grafikprogrammierung erläutert werden.

Das Buch "Procedural Elements for Computer Graphics" kann man wohl schon fast als Klassiker oder Standardwerk auf dem Sektor betrachten. Kaum ein Literaturverzeichnis aus diesem Themenkreis, in dem dieses Buch nicht erwähnt würde. Es ist im wesentlichen eine Algorithmensammlung, die die wichtigsten Verfahren und Techniken gerade auch für die in Einführungen meist sehr kurz behandelten komplexeren Probleme wie Rendering, Transparenz, Schatten und Lichteffekte sehr gut erklärt, jeweils

abgerundet durch ausführliche Literaturhinweise.
Leider kommt man, der Komplexität der Themen wegen, nicht um sehr gute mathematische Kenntnisse, vor allem der Vektor- und Matrixrechnung, aber auch der Analysis herum. Auch Englisch sollte man, wie schon für David Rogers ersten Band, flüssig lesen können.
Die Besonderheit des Buches wird deutlich, wenn man bemerkt, daß über zehn (!) verschiedene Lösungen allein zum Problem verdeckter Oberflächen vorgestellt werden. Besonders hervorzuheben ist auch die sehr gründliche Einführung in den Themenkreis Licht und Farbe, der für realistische Computerbilder extrem wichtig ist.
Vier Seiten mit Farbbildern, viele Schwarz-Weiß-Abbildungen und viele Flußdiagramme bzw. Pseudocode-Implementierungen (eine Art Programmiersprache, die sich leicht in alle gängigen Sprachen übersetzen läßt) runden das Buch ab, das mit 36,40 DM auch nicht allzu teuer ist.
Dieses Buch muß man einfach haben, wenn man ernsthaft an der Materie interessiert ist.
Wer nicht ganz so viel über die Theorie wissen will, sondern lieber den Aufbau einer vollständigen Implementierung eines GKS-ähnlichen 3D-Grafiksystems verfolgt, wird sich voller Begeisterung Steven Harringtons "Computer Graphics - A Programming Approach" hingeben. In diesem Taschenbuch, das übrigens aus der gleichen Serie wie die Bücher von David F. Rogers stammt, wird ein komplettes 3D-Grafiksystem, beginnend mit Algorithmen zum Zeichnen von Linien, implementiert. Selbst Schattierungsfunktionen werden eingebaut.
Dabei ist das Buch alles andere als eine Sammlung von Listings, im Gegenteil: Die implementierten Funktionen samt notwendiger Mathematik werden aufs genaueste erklärt. Dabei ist der Stil locker und flüssig, das Buch liest sich sehr gut. Englisch muß man natürlich trotzdem können... Die Voraussetzungen, die die Hardware erfüllen muß, sind relativ gering. Lediglich eine Routine, die einen Bildschirm- oder Plotterpunkt setzt, muß zur Verfügung stehen, da sehr sauber und geräteunabhängig programmiert wurde.
Geschrieben sind die einzelnen Module in einer Pseudo-Programmiersprache, die an Algol angelehnt ist. Die Übersetzung in eine andere Sprache ist völlig problemlos, vorausgesetzt, Prozeduraufrufe stehen zur Verfügung. Davon kann man ja selbst in den meisten Basic-Varianten ausgehen.
Der Autor legt Wert auf die Feststellung, daß es ihm nicht darum ging, ein besonders effizientes Grafiksystem zu schreiben, sondern grundsätzliche Prinzipien aufzuzeigen. Das Ergebnis ist eine GKS-ähnliche 3D-Grafikbibliothek, die sich sehen lassen kann. Das ganze (ungefähr 4000 Zeilen) erfordert zwar viel Tipparbeit, aber selbst wenn man sich diese Arbeit nicht machen will: Das Buch ist sehr empfehlenswert.
Ebenfalls bei McGraw-Hill erscheint eine Bücherserie im DIN A4-Format namens 'Schaum Überblicke und Aufgaben'. Diese Bücher enthalten zusätzlich zum eigentlichen Stoff zahlreiche Aufgaben und Lösungen, sind also richtige (Selbst-) Lehrbücher. Auch zur Computergrafik ist ein solcher Band erschienen: "Computergrafik - 442 Aufgaben und Lösungen" von Roy A. Plastock und Gordon Kalley (39,50 DM). Wie Sie bereits aus dem Titel erkennen können, handelt es sich hierbei um eine deutsche Übersetzung, deren Stil allerdings extrem trocken geraten ist.
Es ist auch weniger der eigentliche Textteil, der es dem Autor angetan hat. Die Einführung in die Thematik ist in einigen anderen Werken besser gelungen. Sehr gut aber sind die Aufgaben gestellt; vor allem sind sie mit ausführlich erklärten Lösungen versehen. Das ist eine hervorragende Verständniskontrolle, auch wenn es manchmal an Schule erinnert. Dieses Buch kann man nicht lesen, man muß es durcharbeiten. Das ist aber sicherlich sehr effizient.
Eine weitere verhältnismäßig billige und vor allem deutsche Einführung bietet Ian O. Angells Buch "Graphische Datenverarbeitung", das im Hanser-Verlag erschienen ist. Leider sind die zahlreichen Programmbeispiele in diesem Buch in Fortran geschrieben, so daß einige Vorkenntnisse in dieser Sprache vorhanden sein sollten. Die Erklärungen sind aber gut und leicht verständlich. Das Buch beschäftigt sich ausführlich mit der Mathematik für zwei- und dreidimensionale Computergrafik, ohne Vorkenntnisse geht es aber trotzdem auf keinen Fall.
Wer ständig auf dem laufenden über die neuesten Entwicklungen der Branche sein möchte, muß über einen großen Geldüberschuß verfügen. Die im Berliner Springer-Verlag erscheinenden Berichte (Aufsatzsammlungen) über diverse Computergrafik-Konferenzen sind ausnahmslos hochinteressant und sauber gebunden, aber leider auch sehr teuer. Als Beispiel sei der gerade erschienene Bericht über die CG International in Tokio, "Computer Graphics 1987" genannt, der eine Fülle wirklich aktueller Beiträge enthält, so zum Beispiel über ein extrem schnelles Ray Tracing -Verfahren. Dieses nicht einmal besonders dicke Buch kostet leider 180.- DM.
Ein anderes, besonders interessantes Werk aus der Reihe Eurographic Seminars trägt den Titel "Advances in Computer Graphics" und enthält neben neuen Beiträgen auch sehr gute Zusammenfassungen von Grundlagenwissen.
Auch von anderen Vereinigungen und Konferenzen lassen sich die jeweiligen Konferenzunterlagen beschaffen, so zum Beispiel die SIGGRAPH-Tutorials (wichtigste Computergrafik-Messe und Konferenz) oder die "ACM Transactions on Graphics". Leider sind diese Dokumente auch für Mitglieder der jeweiligen Gruppe recht teuer; die ACM Transactions kosten beispielsweise für Mitglieder vierteljährlich 24 $, für Nichtmitglieder aber 65 $. Wenn Sie eine dieser Publikationen beziehen wollen, wenden sie sich am besten an eine Fachbuchhandlung. Die wichtigsten Veröffentlichungen sind:

1) ACM Transactions on Graphics
2) Computer Graphics (SIGGRAPH-Special Interest Group on Graphics)
3) Computer Graphics and Image Processing
4) IEEE Computer Graphics and Applications

VON JOBST HERMEYER

GRAPHTALE PFLANZEN

ODER: DAS ENDE DES DIGITALEN BAUMSTERBENS

Auf immer mehr Schirmen (und Monitoren) flimmern dem Betrachter künstliche Realitäten entgegen. Vektorale Fahrzeuge transportieren ihre imaginären Passagiere über fraktal erzeugte Gebirge. Doch zu einer wirklichkeitsnahen Landschaftssimulation gehören selbst bei der Spielzeugeisenbahn Bäume und Pflanzen; und wie dem Landschaftsmaler stellt sich dem Computerfreak die Frage nach Verfahren zur Abbildung der unendlichen Vielfalt natürlicher Erscheinungsformen dieser Lebensart.

Eine nützliche Beschreibung solcher Algorithmen fand ich in einer Ausgabe der Zeitschrift "Spektrum der Wissenschaft"[1]. Die ursprüngliche Idee stammt dabei von Alvy Smith, dem Leiter des Bereichs Forschung und Entwicklung bei der kalifornischen Firma PIXAR, die Computergrafikfans hinlänglich durch ihre Supercomputer für Trickgrafiken bekannt ist.

Körperbau

Als erstes gilt es, sich über die Anatomie einer Pflanze oder von Pflanzen im Allgemeinen klar zu werden. Der absolute Nichtbotaniker betrachtet zum Beispiel einen Baum als dicken Stamm aus dem Äste kommen, aus welchen Äste kommen an denen Blätter hängen. Zu "Ästen aus Ästen .." fällt dem versierten Informatiker als erstes Stichwort "Rekursion" ein, doch dazu später.

Ein weiteres optisches Merkmal ist die Unregelmäßigkeit, mit der die Äste scheinbar in alle Richtungen wachsen, die den Baum als Ganzes jedoch symmetrisch erscheinen lassen. Um dieser Tatsache Rechnung zu tragen, beziehungsweise mit dem Problem des stufenweisen Wachsens von Pflanzen fertigzuwerden, fand Smith eine passende Methode.

Grafische Graphen

Sie besteht daraus, die Pflanze als Graphen zu repräsentieren, der durch eine bestimmte Bildungsregel zustande kommt. Ein Graph ist ein mathematisches Gebilde (Struktur), welches sich aus Punkten (Vertices) und aus Verbindungen (Kanten,Bögen) zwischen diesen zusammensetzt [2] [3]. In der Entscheidungstheorie und zum Beispiel bei Sortierverfahren tritt eine bestimmte Art von Graphen auf : die Bäume. Gesucht, gefunden ? Leider nein, denn diese Sorte von Bäumen ist meistens etwas zu regelmäßig (besonders die binären) und hat vor allem keinen Stamm. Man könnte sie höchstens zu Darstellung von Büschen verwenden.

Aber um Mißverständnissen vorzubeugen : es soll hier die bildliche Repräsentation eines Graphen zur Abbildung einer Pflanze verwendet werden und nicht der Graph selbst, da mathematische Gebilde stets körperlos sind. Und auch diese Abbildung muß man sich in der Regel zurechtschneidern; Graph-Bäume zum Beispiel werden meistens mit ihrer "Wurzel" oben dargestellt, echte haben letztere aber, der Schwerkraft gehorchend, im Boden verankert.

Um realistische Bilder von Pflanzen entstehen zu lassen, braucht man also eine spezielle Bildungsregel für Graphen. Die dadurch erzeugten Gebilde müssen bestimmten Kriterien gehorchen; sie sollten zum Beispiel eine Art Stamm aufweisen und zu einem "Wachstum zur Krone" tendieren.

L-SYSTEME

Hier kommen die sogenannten L-Systeme zur Hilfe. L-Systeme sind eine Art von Grammatiken, mit der aus einer gegebenen Zeichenkette und einem Regelwerk eine neue Symbolkombination geschaffen werden kann. Sie wurden 1968 durch die dänische Biologin und Mathematikerin Astrid Lindemeyer eingeführt.

Das Handwerkszeug zur Benutznung eines L-Systems ist denkbar einfach. Es besteht aus
a) einem Zeichensatz, der in diesem Fall vier Zeichen hat : [,] , I , *;
b) einem Regelsatz, der die Umsetzung jedes Teils des Zeichensatzes in eine Zeichenkette mit ein oder mehreren Zeichen bewirkt:

```
[ -> [
] -> ]
I -> I I
* -> I [ * ] I [ * ] *
```

Eine Operation besteht dann daraus, eine beliebige Eingangskette in eine Ausgangskette mutieren zu lassen, in dem man jedes Zeichen durch die vorgeschriebene Umsetzung austauscht. Schickt man eine “Saat” in Form eines * durch den Wachstumskanal, erhält man am anderen Ende schon einen Sprößling in Form von “I[*]I[*]*”. Diese Zeichenkette dient der nächsten Mutationsstufe als Eingang und ergibt bereits beachtliche “II[I[*]I[*]*]II[I[*]I[*]*]I[*]I[*]*”, eine Liste mit immerhin 35 Zeichen. Die dritte Wachstumsphase, die diesen Wurm zu bearbeiten hat, liefert am anderen Ende 117 Zeichen ab.
Die Umsetzung dieses Verfahrens in einen Algorithmus führt nun zu der eingangs erwähnten Rekursion. Rekursion bedeutet im mathematischen Sinn die Bestimmung einer Größe aus einer unmittelbar vorausgehenden gleichartigen Zahl oder Funktion. Anders gesagt ist jeder Prozeß, der zur Bestimmung einer beliebigen Prozeßstufe von sich selbst in einer früheren Phase Gebrauch macht, rekursiv (z. B. : X[t] = X[t-1] + 5). Auf die L-Systeme bezogen kann man sagen, daß jede Mutationsstufe durch die Anwendung der vorgegebenen Grammatik auf die vorhergehende Stufe erzeugt werden kann. Prädestiniert für die Umsetzung solcher Problemstellungen ist die dem Bereich der künstlichen Intelligenz zugeordnete Programmiersprache LISP. Sie wurde deshalb zur Implementierung einer Musterlösung für die L-Systeme gewählt (eine Version von XLISP ist übrigens im Public Domain Service der Kickstartredaktion erhältlich). Beim zweiten Daraufschauen auf Listing 1, nachdem man sich erst einmal durch das Gewirr von Klammern durchgefunden hat, die allerdings ein ausgesprochenes Charaktermerkmal von LISP sind, sollte man auch ohne größere Kenntnis der Sprache die Wirkungsweise des Algorithmus durchschauen. Als Hilfestellung für in LISP ungeübte Leser sei vielleicht gesagt, daß die Funktion CAR das erste Element einer Liste liefert, während CDR die Liste mit allen Elementen außer dem ersten zurückgibt.
Um etwa die dritte Stufe eines L-Systems zu erzeugen, würde der Aufruf mit dem LISP-Programm so aussehen:

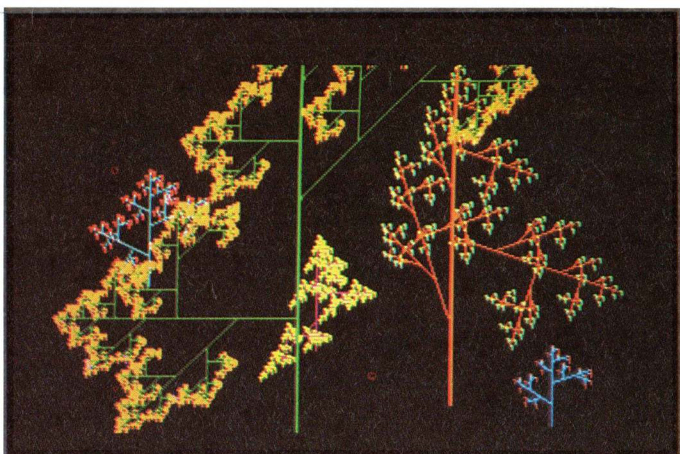


Digitales Laub wird niemals welk.

(L-SYSTEM(L-SYSTEM(L-SYSTEM('(*)))).
Das Ergebnis ist die schon oben beschriebene Liste mit 117 Einträgen.
So weit, so gut; aber selbst 1000 solcher Zeichen haben noch nicht die entfernteste Ähnlichkeit mit einer Pflanze. Das können sie auch nicht, denn die Zeichen sind nur die Bauanleitung zur gesuchten Darstellung des Objekts.

Bauplan

Genauso wie die DNS einer Körperzelle nur eine Beschreibung zum Körperbau liefert, steht auch jeder Eintrag des Zeichensatzes für eine bestimmte Transkriptionsregel. Diese Übersetzungsregeln müssen anstatt von Anweisungen zur Zellkonstruktion Handlungsbeschreibungen zum Zeichnen eines Gebildes in einem zweidimensionalen Raum enthalten. Dabei ist es zunächst gleichgültig, wer diese Anweisungen ausführt; das könnte statt einem Computer etwa auch ein mit Lineal und Stift gewappneter Mensch sein. Wichtig ist aber, im Vorfeld die Rahmenbedingungen zu klären. Diese Voraussetzungen sind:
- es gibt zwei Raumachsen (X-Achse = horizontale Sicht, Y-Achse = vertikale Sicht).
- Punkte lassen sich durch beliebige Koordinatenpaare x,y beschreiben.
- es können Variablen als Merker beliebig eingeführt werden.
Mit Hilfe solcher expliziten Annahmen und der Vorgabe,1 auftretende implizite Vereinbarungen als Definitionen gelten zu lassen, können die benötigten Transkriptionsregeln folgend formu-liert werden:
Das Zeichen I = Zweigstück bedeutet:



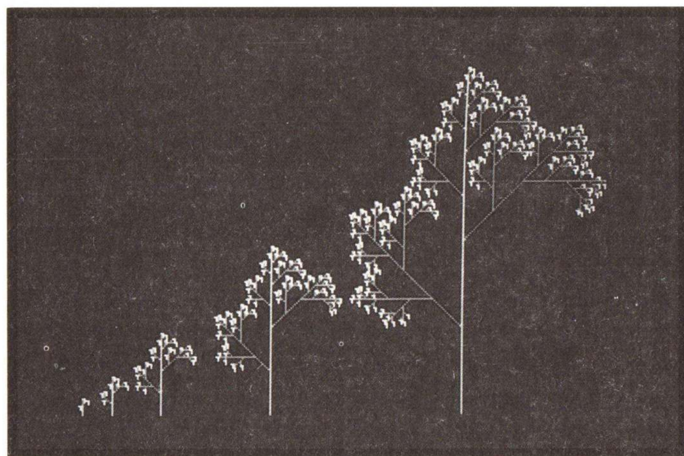
Variationen.

Zeichne vom Koordinatenpaar xs , ys eine Linie der Länge L im Winkel w; weise xs , ys die Endpunkte der gezeichneten Linie zu (xs=xe; ys=ye).

Für das Zeichen * = Zweigende mit Blatt gilt:
Verfahre wie bei einem I. Zeichne an den Endpunkt der Linie (Koordinaten xe,ye) ein Blatt in beliebiger Form.

[= Start einer Abzweigung:
Sichere das Koordinatenpaar xs,ys in xm,ym (xm=xs,ym=ys) und den Winkel w in wm (wm=w). Ist die logische Variable RICHTUNG wahr, dann führe die Operation w = w + d aus; anderenfalls führe w = w - d aus. Negiere den Wert der Variablen RICHTUNG (RICHTUNG = NOT(RICHTUNG)).

] = Ende einer Abzweigung:
Weise dem Koordinatenpaar xs,ys die gesicherten Werte zu (xs=xm,ys=ym); weise dem Winkel w den dafür gesicherten Wert zu (w=wn).



Fünf verschiedene Stufen der gleichen Pflanze.

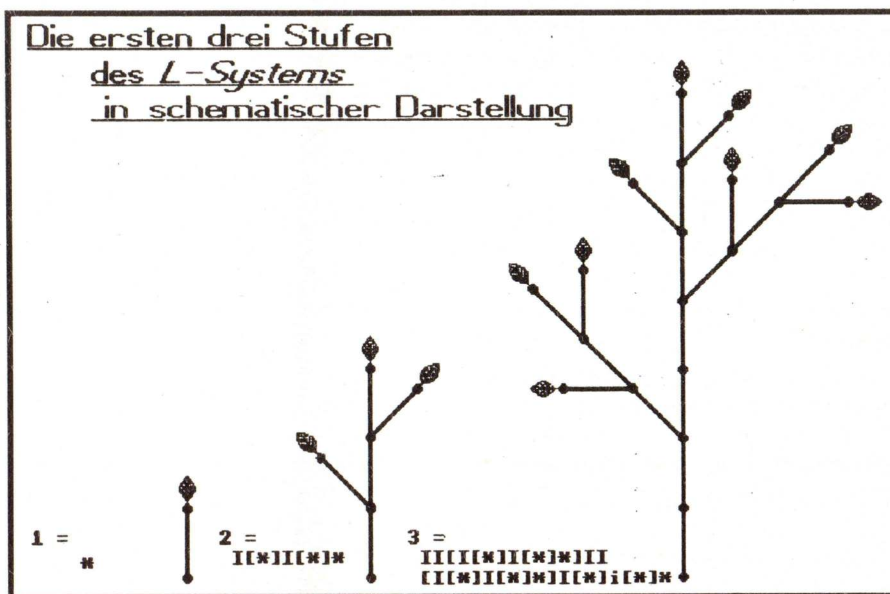
Die schematische Anwendung dieser Regeln kann man sich in Abbildung 2 verdeutlichen. Der Anfangswert für den Winkel w und das Winkelinkrement d sind hier jeweils mit 45 Grad angenommen. Durch die als Schalter wirkende Variable RICHTUNG wird jeder Ast gegenüber seinem Vorgänger um 90 Grad versetzt. Und da bei der ganzen Sache immer wieder die gleichen Regeln verwandt werden, ist eine Prozedur, die eine L-Systemkette in dieser Weise bearbeitet, vorzugsweise rekursiv zu gestalten.

Das Programm

Das zugehörige Programm wurde ursprünglich in C geschrieben und ist für die KICKSTART in Modula-2 umgewandelt worden. Das abgedruckte Listing (Nr. 3) ist in M+V-Modula, dem Compiler eines schweizer Softwareherstellers, implementiert. Ziemlich in der Mitte des Listings finden sich die Prozeduren LSYSTEM, Blatt und ZeichnePflanze, die Kernstücke des Programms. LSYSTEM ist "einfach rekursiv" (single tailrecursive [4]), das heißt, daß die Prozedur sich selbst nur einmal aufruft und daß die Rückgabewerte eigenberechnet oder von einem Rekursionsergebnis abhängig sind. Bei ZeichnePflanze taucht die Rekursion bei der Bearbeitung einer Abzweigung auf. Dort startet sie sich selbst mit den neuen Werten für einen Ast.

Gegenüber den oben beschriebenen Grundregeln arbeitet das Programm aber mit erweiterten Spezifikationen. Die variablen Komponenten einer zu zeichnenden Pflanze lassen sich leicht an der RECORD-Beschreibung "Pflanzenaufbau" ablesen. Mit diesen Teilen und natürlich den dazugehörigen Initialisierungen von Fenster, Schirm und Farben könnte man schon eine Pflanze vollständig auf den Schirm bringen. Da das Festlegen der Variablenwerte im Programm aber zeitraubendes Recompilieren und "Linken" erfordert, wurde die Eingabe der Daten für die Pflanzen auf eine externe Datei verlegt. Zur Bedienungsbequemlichkeit enthält das Programm auch noch ein kleines Menüsystem, mit dem der Vorgang der Pflanzendarstellung wiederholt oder das Programm beendet werden kann. Zu diesem gesamten Komplex lohnt sich sicher ein Blick in den Quellcode, da Modula-2 die Erstellung informativer Programme von Haus aus fördert. An dieser Stelle soll auch einmal auf die Unterschiede der Programmierung in C und in Modula-2 auf dem Amiga eingegangen werden. Modula-2 ist vom Standpunkt der übersichtlichen, sicheren Programmgestaltung C bestimmt vorzuziehen, da es zum Beispiel umfangreiche Typ- und Bereichsprüfungen aufweist. Die vorliegende Version ist offensichtlich aber nicht besonders effizient, weil das ausführbare Element des Programms dreimal so lang und leider auch dreimal so langsam wie das mit Aztec-C bearbeitete ist. Zugeständnisse muß man auch bei der Programmierung machen, wenn es darum geht, die Amigafeatures zu nutzen. Die elegante Art, zum Beispiel Screens gleich bei der Definition zu initialisieren, entfällt gegenüber C, da bei Modula eine normale Zuweisung erfolgen muß. Mithin muß der Programmierer jeden Komponentennamen kennen, ein Umstand, der auf anderen Seite wieder zur besseren Dokumentation der Programme beiträgt. Zudem weist Modula-2 wie PASCAL die wertvolle Anweisung WITH auf, mit der man sich wenigstens das volle Ausschreiben der Strukturkomponentennamen ersparen kann. Was die restlichen Amigatools, wie etwa die Grafikbefehle angeht, kann Modula-2 leider auch nicht entzücken. Durch die in weiten Teilen zeigerorientierte Auslegung der Amiga-Betriebssystemroutinen kann nämlich auch die ausführliche Typenprüfung nicht immer vor GURUs schützen. Es kann eben nicht automatisch sichergestellt werden, daß ein Zeiger den richtigen beziehungsweise überhaupt einen Wert enthält.

Für Leser, die das Programm zurück nach C versetzen wollen, gilt es als erstes, die Unterbereichsvereinbarungen wieder in "Integers" umzuwandeln. Das ist dann auch eine Stärke von Modula-2, auf die man in C leider zu verzichten hat. Es ist nun einmal



viel verständlicher, "von Rot bis Blau" zu sagen, anstatt "von 0 bis 2". Ansonsten sind die Befehle an sich recht ähnlich, insbesondere aber die Amigafunktionen, die man eigentlich nur abschreiben muß.

Die Benutzung

Als nächstes also zur Anwenderseite des Programms. Sein "Futter" erhält es durch eine normale ASCII-Datei. Dort trägt man die Werte, etwa mit Komma getrennt, mit Hilfe eines Editors ein. Ein Semikolon oder Ausrufezeichen bedeutet, daß der Rest der jeweiligen Zeile als Kommentar betrachtet wird. Davon abgesehen müssen alle Werte in einer bestimmten Reihenfolge erscheinen. Die Vereinbarung dafür ist:

a) Wachstumsstufe

Sie kann im Bereich zwischen 1 und 8 liegen und gibt an, wie oft die L-Systemkette, startend mit einem *, mutiert wird. Man sollte allerdings bedenken, daß eine größere Mutationsstufe mehr Speicher benötigt und länger dauert. Für eine Pflanze der Wachstumsstufe 7 zum Beispiel erhöhte ich den Stack auf 300 KBytes und mußte über eine halbe Stunde auf das Ergebnis warten.

b) Verdickter Stamm

Eine 0 steht für falsch und heißt, daß der Stamm, wie die Äste, nur einen Pixel breit ist. Eine 1 bedeutet einen doppelt so breiten Stamm. Als Stamm wird übrigens jeder vertikale Strich angesehen, der auf der X-Koordinate der Wurzel liegt (siehe c).

c)X-/Y-Wurzel

Dieses Koordinatenpaar gibt den Ursprung der Pflanze auf dem Bildschirm an. Da mit der "kleinen" Auflösung gearbeitet wird, darf X für den PAL-Schirm bis 256 und Y bis 320 gehen. Der Koordinatenursprung für diese Angaben befindet sich in der linken oberen Ecke des Bildschirms.

d) Wurzelneigung

...sollte in dieser Programmversion immer 90 sein, da sonst der Stamm verschwindet (kann übrigens auch gewünscht sein). Dieser Wert trägt bei Abweichung von 90 Grad zur Anfangsneigung der Äste bei. Erlaubte Zahlen liegen zwischen 0 und 360.

e) Linker und rechter Astwinkel

Sie bestimmen die Neigung der Pflanzenäste in je eine Richtung und können im Bereich von 0 bis 360 liegen.

f) Segmenthöhe

...legt die Höhe für jedes einzelne Segment der Pflanze fest und sollte nicht größer als 30 werden. Generell gilt: Je höher die Wachstumsstufe (Wert aus a), desto kleiner muß die Segmenthöhe sein, damit die Pflanze noch auf den Bildschirm paßt. Als Regel läßt sich auch formulieren, daß die Segmenthöhe kleiner (256/ (2 hoch Wachstumstufe)) sein sollte. 256 ist die jetzige maximale Bildschirmhöhe.

g) Farbenwerte

Es folgen nun drei Wertequadrupel. Die jeweils erste Zahl einer Viererserie ist die Farbnummer im Bereich 1 - 31. Die anderen drei repräsentieren die Rot-, Grün-, Blaukomponenten dieser Farbe. Sie können zwischen 0 und 15 liegen. Die erste dieser Wertekombinationen ist für die Farbe des Stamms zuständig, die zweite für die Äste und die dritte für die Blätter. Das Schema lautet demnach:

Stamm -> Farbennummer, Rot,Grün, Blau
Äste -> Farbennummer, Rot, Grün, Blau
Blätter -> Farbennummer, Rot, Grün, Blau.

Diese Farbzahlen werden im Programm über RGB4()-Anweisungen als Farbregister initialisiert. Die Farbzahlenaufzählung in dieser Weise ist zugegebenermaßen nicht sehr rationell. Will man nämlich eine Farbe mehrmals verwenden, müssen alle dazu nötigen Angaben an der gewünschten Stelle wiederholt werden.

Im Allgemeinen kann diese Datendatei beliebig viele hintereinanderstehende Pflanzen enthalten, die Angaben für jede einzelne müssen jedoch vollständig sein. Ein Beispiel einer Parameterdatei mit zugehörigem Ergebnis zeigen Listing 2 und Abbildung 4. Dort habe ich den Pflanzen Namen gegeben, woran man recht gut die Benutzung von Kommentaren ablesen kann. Hat man die Datei fertig erstellt, erfolgt der Programmaufruf durch: Pflanze Parameterdatei. Vor dem Programmaufruf sollte auf jeden Fall der Stack erhöht werden (für die Datei in Listing 2 kommt man mit 70000 Bytes aus), da die rekursive Programmierung natürlich extrem speicherintensiv ist.

Horizonte

Die vorliegende Implementation der graphtalen Pflanzen will beileibe keine vollständige Anwendung sein. Da ich gerade das Modulaprogramm aus Zeitdruck nur in seinen Hauptfunktionen getestet habe, können sich durchaus noch kleinere Bugs finden. Ich bitte, das zu verzeihen und auftretende Fehler der Redaktion mitzuteilen.

Außerdem ist das Output im Moment wohl eher von künstlerischem als von Realitätswert. Das heißt aber auch, daß die graphtalen Pflanzen damit noch lange nicht ausgereizt sind und unterstreicht etwas den experimentellen Charakter der ganzen Sache. Veränderungsmöglichkeiten sind deshalb auch reichlich vorhanden. So setzt sich etwa ein Blatt einfach aus vier Pixeln einer Farbe zusammen. Das ist natürlich leicht zu ändern, wenn man bereit ist, über die Veränderungen des Parameterfiles hinaus in das Programm einzugreifen. Bizzarste Abarten kann man dann auch durch das Verändern der L-Systemregeln erzeugen. Weiterhin läßt sich mit der Rechenungenauigkeit spielen, indem man die Konstante "Rundungsfaktor" in der Prozedur AbsRound modifiziert. Auch kann man den Stamm noch in verschiedenster Weise manipulieren. Die Krönung wäre natürlich, statt der direkten Verbindung zwischen Segmenten mittels Move() und Draw() Kreisbögen verschiedener Winkel zu benutzen. Wie man sieht, stehen noch alle Möglichkeiten offen, sich in kreativer Programmierung kräftig zu üben.

Literaturhinweise:

[1] Spektrum der Wissenschaft März 1987 Spektrum der Wissenschaft Verlagsgesellschaft mbH & Co, Heidelberg

[2] Graph Algorithms, Simon Even Computer Science Press, Rockville USA , 1979

[3] Lehrbuch der Mathematik für Wirtschaftswissenschaftler Westdeutscher Verlag , Opladen, 1975

[4] Lisp, P. H. Winston, B. K. Horn, Addison-Wesley Publishing Company, 1984

```
                    Listing 1
                Das L-System in LISP

    (DEFUN L-SYSTEM (START-LIST)
      (COND
      ((NULL START-LIST) NIL)
        ((EQUAL (CAR START-LIST) '*)
        (APPEND  '(I [ * ] I [ * ] *) (L-SYSTEM (CDR START-LIST))))
       ((EQUAL (CAR START-LIST) 'I)
        (APPEND  '(I I) (L-SYSTEM (CDR START-LIST))))
       ((EQUAL (CAR START-LIST) '[)
        (APPEND  '([) (L-SYSTEM (CDR START-LIST))))
       ((EQUAL (CAR START-LIST) '])
        (APPEND  '(]) (L-SYSTEM (CDR START-LIST))))))
```

```
;                   Listing 2
; Parameterdatei fuer verschiedene Pflanzen zur Erzeugung mit
; dem Programm Pflanze
;
; serengeti
    5,         ; Wachstumsstufe
    1,         ; Verdickter Stamm (1=Ja, 0=Nein)
    140,       ; Wurzel X Ursprung
    240,       ; Wurzel Y Ursprung
    90,        ; Wurzelneigung
    -15,       ; Linker Astwinkel
    35,        ; Rechter Astwinkel
    4,         ; Segmenthoehe
    2,3,10,3,  ; FarbNummer, Rot- Gruen- und Blaukomponente fuer den Stamm
    1,1,12,0,  ; FarbNummer, Rot- Gruen- und Blaukomponente fuer Aeste
    3,12,7,0   ; FarbNummer, Rot- Gruen- und Blaukomponente fuer Blaetter
; norm
 5,1,40,190,
 90,-45,45,3,
    4,3,12,4,
    5,1,12,0,
    6,7,7,12
; dixi
 6,1,240,180,
 90,-60,20,1,
    7,10,2,2,
    8,13,2,0,
    9,14,14,0
; little
 3,0,280,195,
 90,-40,40,5,
    3,12,7,0,
    6,7,7,12,
    2,3,10,3
; tinman
 4,0,248,200,
 114,-13,32,7,
    10,10,10,10,
    11,13,13,13,
    12,3,10,8
```

```
(*                         Listing 3                                  *)
MODULE pflanze;
(* Programm zu Erzeugung graphtaler Pflanzer aus einer Datendatei
   lauffaehig compiliert mit Amiga Modula-2 Version 3.1d von AMSoft
              Author : -=JH=- *)
FROM Arguments IMPORT
 NumArgs,GetArg;
FROM ASCII IMPORT
 eol;
FROM FileMessage IMPORT
 StrPtr,ResponseText;
FROM FileSystem IMPORT
 Response,File,Lookup,Close,ReadChar;
FROM Strings IMPORT
 Length,Insert;
FROM InOut IMPORT
 WriteLn,WriteString,WriteInt,WriteCard,Write;
FROM SYSTEM IMPORT
 ADDRESS,ADR,SHIFT,LONGSET;
FROM Arts IMPORT
 Assert;
FROM Graphics IMPORT
 jam1, jam2, FontFlagSet, FontStyleSet, RastPortPtr, TextAttr, ViewModes,
 ViewModeSet, ViewPortPtr,
 Draw, Move, RectFill, SetAPen, SetBPen, SetDrMd, SetRGB4,
 WritePixel,ClearScreen;
FROM Intuition IMPORT
 stdScreenHeight, customScreen, IDCMPFlags, IDCMPFlagSet, IntuiMessagePtr,
 NewScreen, NewWindow, ScreenPtr, WindowFlags, WindowFlagSet, WindowPtr,
 Menu, MenuItem, IntuiText, MenuItemFlags, MenuItemFlagSet,
 menuNull, ClearMenuStrip, SetMenuStrip,menuEnabled,
 CloseScreen, CloseWindow, OpenScreen, OpenWindow;
FROM Exec IMPORT
 MessagePtr, WaitPort, GetMsg, ReplyMsg;
FROM MathLib0 IMPORT
 pi,sin,cos;

CONST
MaxWachstumsstufe=8;
MaxVerdickterStamm=1;
MaxWurzelX=320;
MaxWurzelY=256;
MaxWinkel=360.0;
MaxSegmenthoehe=30.0;
MaxFarbenNr=30;
MaxFarbenanteil=15;

TYPE
Grundfarben=(rot,gruen,blau);
Pflanzenteile=(stamm,zweig,blatt);
HauptMenus=(pflanzenmenu);
Pflanzenmenus=(neuzeichnen,ende);
Pflanzenaufbau =
 RECORD
     Wachstumsstufe : CARDINAL;
     VerdickterStamm : BOOLEAN;
     WurzelX,WurzelY : INTEGER;
     Wurzelneigung,
     AstWinkelLinks,AstWinkelRechts : REAL;
     Segmenthoehe : REAL;
     FarbenNr : ARRAY Pflanzenteile OF CARDINAL;
     Farbenanteil : ARRAY Pflanzenteile OF
            ARRAY Grundfarben OF CARDINAL;
  END;

VAR
 s: ScreenPtr;
 w : WindowPtr;
 rp : RastPortPtr;
```

```
 vp : ViewPortPtr;
 LKETTE : ARRAY[0..10000] OF CHAR;
 Pflanze : Pflanzenaufbau;
 MenuBalken : Menu;
 UnterTeile : ARRAY [1..2] OF MenuItem;
 MenuTexte : ARRAY [1..4] OF IntuiText;
 MsgPtr: IntuiMessagePtr;
 finish: BOOLEAN;
 class : IDCMPFlagSet;
 code  : CARDINAL;
 Ende,MenuOK : BOOLEAN;
 Filename : ARRAY[1..50] OF CHAR;
 FilenameL : INTEGER;

PROCEDURE AbsRound(Input : REAL) : INTEGER;
CONST
Rundungsfaktor=0.5;
VAR
TempInt :INTEGER;
BEGIN
TempInt:=TRUNC(ABS(Input)+Rundungsfaktor);
IF Input < 0.0 THEN TempInt:=TempInt*(-1); END;
RETURN TempInt;
END AbsRound;

PROCEDURE InitIntuiText(VAR it: IntuiText; text: ADDRESS): ADDRESS;
 BEGIN
  WITH it DO
   frontPen:=0; backPen:=1;
   drawMode:=jam2;
   leftEdge:=0; topEdge:=1;
   iTextFont:=NIL; iText:=text;
   nextText:=NIL
  END;
  RETURN ADR(it)
 END InitIntuiText;

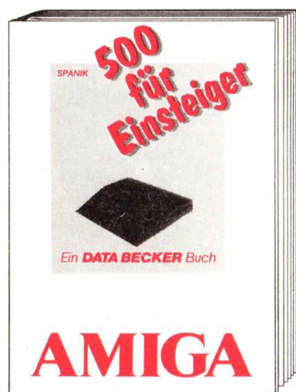

PROCEDURE MenuNUM(code : CARDINAL) : CARDINAL;
BEGIN
 RETURN code MOD 32;
END MenuNUM;


PROCEDURE ItemNUM(code : CARDINAL) : CARDINAL;
BEGIN
 RETURN code DIV 32 MOD 64;
END ItemNUM;

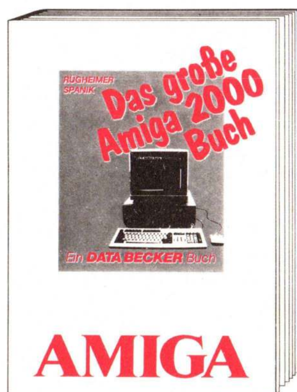

PROCEDURE OPENALL;
 VAR
  ta : TextAttr;
  ns : NewScreen;
  nw : NewWindow;
BEGIN
  ta.name:=ADR('topaz.font');
  ta.ySize:=8; ta.style:=FontStyleSet{}; ta.flags:=FontFlagSet{};

  ns.leftEdge:=0; ns.topEdge:=0;
  ns.width:=320;  ns.height:=255;
  ns.depth:=5;
  ns.detailPen:=0; ns.blockPen:=1;
  ns.type:=customScreen;
  ns.viewModes:=ViewModeSet{};
  ns.font:=ADR(ta);
  ns.defaultTitle:=NIL;
  ns.gadgets:=NIL; ns.customBitMap:=NIL;
  s:=OpenScreen(ns);
```

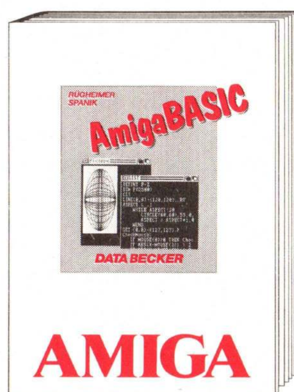
AMIGA BUCHHITS



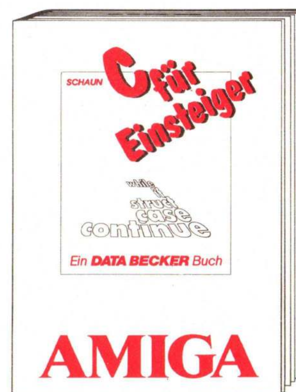
Wählen Sie gleich den richtigen Einstieg zu Ihrem Amiga 500. Denn das Handbuch läßt Sie dabei völlig allein. Versuchen Sie es lieber gleich mit Amiga 500 für Einsteiger. Hier heißt es: anschließen und loslegen. Verständlich für jedermann zeigt Ihnen dieses Buch: Workbench, AmigaBASIC, CLI und AmigaDOS. Locker aufbereitet bietet es Ihnen alles Wissenswerte. Bis hin zu den beim Amiga 500 mitgelieferten Zusatzprogrammen.
Amiga 500 für Einsteiger
343 Seiten, DM 39,–



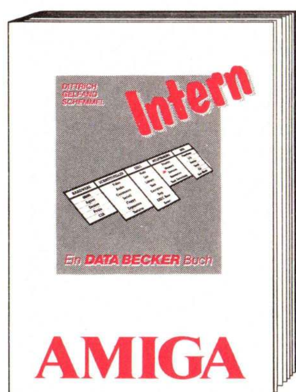
Läßt das Handbuch Sie auch in so manchen Dingen allein, das große Amiga-2000-Buch nicht. Hier finden Sie eine umfassende Einführung in die Arbeit mit Ihrem neuen Rechner – und mehr als das. Sind Sie erst einmal mit dem Amiga 2000 „per Du", zeigen Ihnen die Autoren, was einen Amiga-Profi ausmacht: Kickstart im RAM, PC-Audioausgänge, erste Hilfe bei Harddisk-Abstürzen, Laufwerkeinbau in den Amiga 2000 und, und, und. Sollten Sie also noch Fragen zu Ihrem Rechner haben, hier finden Sie die Antworten.
Das große Amiga-2000-Buch
Hardcover, 684 Seiten, DM 59,–



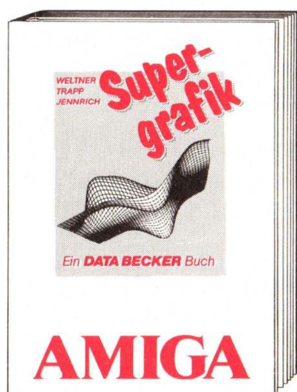
Das erfolgreiche Buch zu Amiga-BASIC. Erweitert um Kickstart 1.2, neuer Workbench und Amiga 500 & 2000. Alles, was BASIC-Programmierern Spaß macht: Grafik und Sound, Laden und Speichern von Graficraft-Bildern in BASIC-Programme, sequentielle und relative Dateien, Business-Grafik, Computeranimation, Windows, Umgang mit IFF-Bildern, Sprachausgabe und, und, und. Das Buch für Einsteiger, Aufsteiger und Profis.
AmigaBASIC
Hardcover, 774 Seiten, DM 59,–



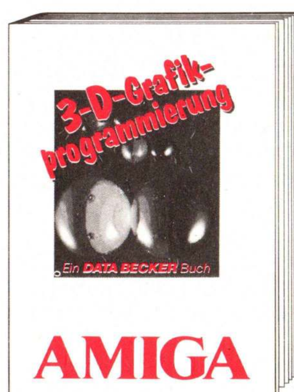
C an einem Wochenende? Durchaus möglich! Mit C für Einsteiger. Ein Einführungskurs, der Ihnen schnell und einfach die wichtigsten Grundlagen dieser Sprache vermittelt. Vom ersten Programm bis hin zu den Routinen in den Bibliotheken. Mit dem gesamten Sprachumfang und den besonderen Features von C. Zahlreiche Tips & Tricks zur Programmierung und eine Beschreibung der beiden Compiler Lattice C und Aztek runden das Ganze ab.
Amiga C für Einsteiger
254 Seiten, DM 39,–



Alles zum Amiga. In einem Band und absolut auf dem neuesten Stand: 68000-Prozessor, CIA, Blitter, Customchips, die wichtigsten Strukturen von EXE, I/O-Handhabung, Verwaltung der Ressources, Multitasking, EXEC-Base, resetfeste Programme, DOS-Funktionen, IFF-Format, Programmstart von CLI und Workbench, Programmierung der EXEC- und DOS-Routinen und, und, und. Eben ein typischer Intern-Band, in dem wieder einmal nichts Wissenswertes fehlt.
Amiga Intern
Hardcover, 639 Seiten, DM 69,–



Keine Frage: Grafik ist das zentrale Thema des Amiga. Hier das entsprechende Know-how, um den Amiga aus der Reserve zu locken: Grafikprogrammierung mit den vorhandenen BASIC-Befehlen, Nutzung der Libraries, die Register der Grafik-Chips, CAD, Aufbau und Programmierung von Screens, Windows, HAM, Halfbrites und Interlace aus BASIC und C. Das Amiga-Buch zum Thema Nr. 1!
Amiga Supergrafik
Hardcover, 686 Seiten, DM 59,–



3-D-Grafikprogrammierung – hier finden Sie Grafikalgorithmen für absolut realistisch gestaltete Bilder. Die einzelnen Vorlagen lassen sich dabei mit einem Editor problemlos eingeben und solange durch die Wahl verschiedener Materialien, Farben, Lichtquellen und Spiegelungen verfeinern, bis sie eine absolut naturgetreue Darstellung erreicht haben.
Amiga 3-D-Grafikprogrammierung
Hardcover, 283 Seiten
inkl. Diskette, DM 59,–

Super

Regelmäßig in der DATA WELT: Amiga Window – das Forum für den engagierten Amiga-Anwender. Mit kreativen Projekten, Interviews, Software-Tests und wichtigen News. Und: Auch was sich sonst so in der Computerszene tut, erfährt der Amiga-Anwender. Die DATA WELT – das aktuelle Computermagazin. Monat für Monat überall da, wo es Zeitschriften gibt.

BESTELL-COUPON

Einsenden an: DATA BECKER · Merowingerstr. 30 · 4000 Düsseldorf 1
Bitte senden Sie mir:

zzgl. DM 5,– Versandkosten
unabhängig von der bestellten Stückzahl
☐ per Nachnahme ☐ Verrechnungsscheck liegt bei

Name
Straße
Ort

DATA BECKER
Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

```
   Assert(s#NIL,ADR("Screen konnte nicht eroeffnet werden"));
   vp:=ADR(s^.viewPort);

   nw.leftEdge:=0; nw.topEdge:=0;
   nw.width:=320; nw.height:=255;
   nw.detailPen:=255; nw.blockPen:=255;
   nw.idcmpFlags:=IDCMPFlagSet{mouseButtons, closeWindow,menuPick};
   nw.flags:=WindowFlagSet{activate,borderless,gimmeZeroZero};
   nw.title:=NIL;
   nw.screen:=s;
   nw.type:=customScreen;
   nw.firstGadget:=NIL; nw.checkMark:=NIL; nw.bitMap:=NIL;
   nw.minWidth:=0; nw.minHeight:=0;
   nw.maxWidth:=0; nw.maxHeight:=0;
   w:=OpenWindow(nw);
   Assert(w#NIL, ADR("Kein Window konnte eroeffnet werden"));
   rp:=w^.rPort;
  END OPENALL;


PROCEDURE INITALL;
  BEGIN
   SetRGB4(vp, 0, 00, 00, 00);
   SetRGB4(vp, 1, 15, 00, 00);
   SetRGB4(vp, 2, 00, 15, 00);
   SetRGB4(vp, 3, 00, 00, 15);
   SetRGB4(vp, 4, 15, 15, 15);
   SetDrMd(rp,jam1);
   SetAPen(rp, 1);
   SetBPen(rp, 0);
   WITH MenuBalken DO
    nextMenu:=NIL;
    leftEdge:=0; topEdge:=0; width:=80; height:=10;
    flags:={menuEnabled};
    menuName:=ADR("Pflanze");
    firstItem:=ADR(UnterTeile[1]);
   END;
   WITH UnterTeile[1] DO
    leftEdge:=0; topEdge:=0; width:=100; height:=10;
    flags:=MenuItemFlagSet{itemEnabled, itemText, highItem};
    nextItem:=ADR(UnterTeile[2]);
    itemFill:=InitIntuiText(MenuTexte[1],ADR("neuzeichnen"));
    selectFill:=InitIntuiText(MenuTexte[2],ADR("NEUZEICHNEN"));
    subItem:=NIL;
   END;
  WITH UnterTeile[2] DO
    leftEdge:=0; topEdge:=10; width:=100; height:=10;
    flags:=MenuItemFlagSet{itemEnabled, itemText, highItem};
    nextItem:=NIL;
    itemFill:=InitIntuiText(MenuTexte[3],ADR("ende"));
    selectFill:=InitIntuiText(MenuTexte[4],ADR("ENDE"));
    subItem:=NIL;
   END;
  END INITALL;


PROCEDURE GOODBYE;
  BEGIN
   IF w#NIL THEN CloseWindow(w) END;
   IF s#NIL THEN CloseScreen(s) END;
 END GOODBYE;


 PROCEDURE LSYSTEM(Stufe : CARDINAL; Startsequenz : ARRAY OF CHAR;
                                 VAR Ergebnissequenz : ARRAY OF CHAR);
 VAR
 N :CARDINAL;
 C : CHAR;
 BEGIN
```

```
 Ergebnissequenz:="";
 FOR N := 0 TO Length(Startsequenz)-1 DO
 C:=Startsequenz[N];
  CASE C OF
    "*" : Insert(Ergebnissequenz,Length(Ergebnissequenz),
          "I[*]I[*]*");|
    "I" : Insert(Ergebnissequenz,Length(Ergebnissequenz),
          "II");|
    "[" : Insert(Ergebnissequenz,Length(Ergebnissequenz),
          "[");|
    "]" : Insert(Ergebnissequenz,Length(Ergebnissequenz),
          "]");
   END;
  END;
  IF Stufe > 1 THEN
    LSYSTEM(Stufe-1,Ergebnissequenz,Ergebnissequenz);
  END;
 END LSYSTEM;


PROCEDURE InitPflanzenFarben;
 VAR
  Index : Pflanzenteile;
 BEGIN
 WITH Pflanze DO
  FOR Index:=stamm TO blatt DO
   SetRGB4(vp,FarbenNr[Index],Farbenanteil[Index,rot],
           Farbenanteil[Index,gruen],Farbenanteil[Index,blau]);
  END;
 END;
END InitPflanzenFarben;


PROCEDURE Blatt(StartX,StartY : INTEGER);
VAR
 dummy : LONGINT;
BEGIN
  SetAPen(rp,Pflanze.FarbenNr[blatt]);
  dummy:=WritePixel(rp,StartX,StartY);
  dummy:=WritePixel(rp,StartX+1,StartY);
  dummy:=WritePixel(rp,StartX+1,StartY+1);
  dummy:=WritePixel(rp,StartX+1,StartY+2);
  SetAPen(rp,Pflanze.FarbenNr[zweig]);
END Blatt;


PROCEDURE ZeichnePflanze(N : INTEGER;
                  (* N bezieht sich auf die Postion innerhalb
                     der globalen Variablen LKETTE *)
                  X,Y : INTEGER; Winkel : REAL) : INTEGER;
CONST
Zweigende = "]";
PIdurch180 = pi / 180.0;
TYPE
Richtungen = (rechts,links);
VAR
C : CHAR;
LinksNeigung : BOOLEAN;
StammVersatz : INTEGER;
WinkelNeu : REAL;

BEGIN
LinksNeigung:=FALSE;
WHILE (N<Length(LKETTE)) AND (LKETTE[N]#Zweigende) DO
 C:=LKETTE[N];
 WITH Pflanze DO
 CASE C OF
    "*" :
     Move(rp,WurzelX+X,WurzelY-Y);
```

```
     X:=X+AbsRound(Segmenthoehe*cos(Winkel*PIdurch180));
     Y:=Y+AbsRound(Segmenthoehe*sin(Winkel*PIdurch180));
     Draw(rp,WurzelX+X,WurzelY-Y);
     Blatt(WurzelX+X,WurzelY-Y); |
    "I" :
     Move(rp,WurzelX+X,WurzelY-Y);
     StammVersatz:=0;
     IF X=0 THEN (* wird der Stamm gezeichnet *)
      SetAPen(rp,FarbenNr[stamm]);
      IF VerdickterStamm THEN
       Draw(rp,WurzelX+X,WurzelY-
             (Y+AbsRound(Segmenthoehe*sin(Winkel*PIdurch180))));
       Move(rp,WurzelX+X+1,WurzelY-Y);
       StammVersatz:=1;
      END;
     END;
      X:=X+AbsRound(Segmenthoehe*cos(Winkel*PIdurch180));
      Y:=Y+AbsRound(Segmenthoehe*sin(Winkel*PIdurch180));
      Draw(rp,WurzelX+X+StammVersatz,WurzelY-Y);
      SetAPen(rp,FarbenNr[zweig]); |
    "[" :
     IF LinksNeigung THEN
      IF Winkel=0.0 THEN Winkel:=360.0; END;
      WinkelNeu:=Winkel+AstWinkelLinks;
     ELSE
      IF Winkel>=360.0 THEN Winkel:=0.0; END;
      WinkelNeu:=Winkel+AstWinkelRechts;
     END;
     N:=ZeichnePflanze(N+1,X,Y,WinkelNeu);
     LinksNeigung:=NOT(LinksNeigung); |
 END;
 END;
 INC(N);
END;
RETURN N;
END ZeichnePflanze;


PROCEDURE PerformPflanzenFile(Filename : ARRAY OF CHAR);
CONST
Saat = "*";
VAR
INFile : File;
OK : INTEGER;
Pflanzenzaehler : CARDINAL;
StrPtrv : StrPtr;
FileEnde : BOOLEAN;

PROCEDURE LiesEinePflanze() : BOOLEAN;
CONST
Exponent=10.0;
MaxElement=20;
VAR
Numerisch,Kommentar,Dezimalpunkt,Weiter,
LeseVorgangOK : BOOLEAN;
IZahl,Elementzaehler,Temp : INTEGER;
NachkommaTeil,Zahl,Vorzeichen : REAL;
C : CHAR;

BEGIN
LeseVorgangOK:=FALSE;
Numerisch:=FALSE;Dezimalpunkt:=FALSE;
Kommentar:=FALSE;Weiter:=FALSE;
NachkommaTeil:=1.0; Zahl:=0.0; Vorzeichen:=1.0;
Elementzaehler:=1;
WHILE NOT(Weiter) AND NOT(INFile.eof) DO
ReadChar(INFile,C);
IF Kommentar THEN
 IF C=eol THEN Kommentar:=FALSE; END;
```

Top aktuell:



Amiga Tips & Tricks – jetzt in einer völlig überarbeiteten Neuauflage. Hier verraten Ihnen echte Profis, mit welchen Tricks sie mehr aus dem Amiga holen: Hilfen zur Gestaltung eigener Programme, Tips & Tricks zum AmigaBASIC, Maschinenprogramme in AmigaBASIC, Einsatz von DOS-Routinen, optimierende Hilfsprogramme für AmigaBASIC-Programme, Tips zur Arbeit mit der Workbench, Aufbau der Icons, neue Ein-/Ausgaberoutine. Mit vielen Anregungen, aber auch fertigen Lösungen. Greifen Sie in die Trickkiste, und schon werden Dinge wahr, die Sie nicht für möglich hielten. Ein Buch, das voller Überraschungen steckt.
Amiga Tips & Tricks – die riesige Fundgrube für jeden Amiga-Besitzer.

Amiga Tips & Tricks
Hardcover, 473 Seiten
DM 49,–

Der Amiga macht es einem so leicht wie möglich. Nahezu alles läßt sich problemlos über die Workbench bearbeiten. Wenn Sie jedoch den Mut haben, die komfortable Oberfläche zu verlassen, werden Sie schon sehr bald belohnt – mit einigen Dingen, die Sie dem Amiga bisher nicht zugetraut hätten. Das große Buch zu AmigaDOS hilft Ihnen dabei. Neben einem ausführlichen Einsteigerteil erfahren Sie alles, was Sie bei Ihrer praktischen Arbeit mit dem AmigaDOS wissen sollten: Umlenken der Ein- und Ausgabe, sinnvoller Einsatz des Jokers, Arbeiten mit RAM-Disk und CLI, Batch-Dateien, STARTUP-Sequenz, Multitasking mit dem CLI, Aufbau der CLI-Befehle, Programmierung eigener CLI-Befehle, neue CLI-Befehle in BASIC und C. Dazu ein ausführlicher, gut strukturierter Nachschlageteil. Wer also mit dem AmigaDOS arbeiten möchte, sollte dieses Buch immer in greifbarer Nähe haben.

Das große Buch zu AmigaDOS
Hardcover, 320 Seiten
DM 49,–

Das Buch, das zur Amiga-Floppy keine Frage offenläßt. Hier finden Sie Dinge, die Sie im Handbuch vergeblich suchen werden: Floppy-Operationen unter der Workbench und unter AmigaDOS im CLI, relative und sequentielle Dateien, Aufbau der Diskette, Zugriff über Trackdisk-Device, Track lesen und schreiben, Kodier-und Dekodierroutinen des Betriebssystems… Mit vielen nützlichen Programmen wie z. B. ein Superkopierprogramm oder einen Floppyspeeder. Was Sie wissen müssen, finden Sie hier – vom Einsteiger zum Profi.

Amiga Floppy Buch
Hardcover, ca. 350 Seiten
inkl. Diskette, DM 59,–
erscheint ca. 1/88

DATA BECKER
Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

BESTELL-COUPON
Einsenden an: DATA BECKER · Merowingerstr. 30 · 4000 Düsseldorf 1
Bitte senden Sie mir:

zzgl. DM 5,– Versandkosten
unabhängig von der bestellten Stückzahl
☐ per Nachnahme ☐ Verrechnungsscheck liegt bei

Name
Straße
Ort

```
ELSE
 IF (C=";") OR (C="!") THEN Kommentar:=TRUE;
ELSE
 IF (C="-") AND NOT(Numerisch) THEN
  Vorzeichen:=-1.0; Numerisch:=TRUE;
 ELSE
  IF (C=".") AND NOT(Numerisch) THEN
   Dezimalpunkt:=TRUE; Numerisch:=TRUE;
  ELSE
   IF (C>="0") AND (C<="9") THEN
    IF NOT(Numerisch) THEN Numerisch:=TRUE; END;
    Zahl:=Zahl*Exponent+(FLOAT(ORD(C)-ORD("0"))*NachkommaTeil);
    IF Dezimalpunkt THEN NachkommaTeil:=NachkommaTeil / 10.0; END;
   ELSE
    IF Numerisch THEN
     Zahl:=Zahl*Vorzeichen; IZahl:=TRUNC(Zahl);
     Numerisch:=FALSE;Dezimalpunkt:=FALSE;
     Kommentar:=FALSE;Weiter:=FALSE;
     NachkommaTeil:=1.0; Vorzeichen:=1.0;
     WITH Pflanze DO
     CASE Elementzaehler OF
     1:
      IF (IZahl>0) AND (IZahl<=MaxWachstumsstufe) THEN
       Wachstumsstufe:=IZahl;
       INC(Elementzaehler);
      ELSE
       WriteString("Falsche Wachstumsstufenangabe in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
     2:
      IF (IZahl>=0) AND (IZahl<=MaxVerdickterStamm) THEN
       IF IZahl=0 THEN VerdickterStamm:=FALSE;
       ELSE VerdickterStamm:=TRUE END;
       INC(Elementzaehler);
      ELSE
       WriteString("Falsche Angabe zu verdicktem Stamm (nur 0, oder 1)");
       WriteLn; WriteString("in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
     3:
      IF (IZahl>0) AND (IZahl<=MaxWurzelX) THEN
       WurzelX:=IZahl;
       INC(Elementzaehler);
      ELSE
       WriteString("Falsche X Ursprungsangabe in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
     4:
      IF (IZahl>0) AND (IZahl<=MaxWurzelY) THEN
       WurzelY:=IZahl;
       INC(Elementzaehler);
      ELSE
       WriteString("Falsche Y Ursprungsangabe in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
     5:
      IF (Zahl>0.0) AND (Zahl<=MaxWinkel) THEN
       Wurzelneigung:=Zahl;
       INC(Elementzaehler);
      ELSE
       WriteString("Falsche Ursprungsneigungsangabe in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
     6:
```

```
      IF (Zahl>=(-MaxWinkel)) AND (Zahl<=MaxWinkel) THEN
       AstWinkelLinks:=Zahl;
       INC(Elementzaehler);
      ELSE
       WriteString("Falscher linker Astwinkel in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
     7:
      IF (Zahl>=(-MaxWinkel)) AND (Zahl<=MaxWinkel) THEN
       AstWinkelRechts:=Zahl;
       INC(Elementzaehler);
      ELSE
       WriteString("Falscher rechter Astwinkel in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
     8:
      IF (Zahl>0.0) OR (Zahl<=MaxSegmenthoehe) THEN
       Segmenthoehe:=Zahl;
       INC(Elementzaehler);
      ELSE
       WriteString("Falsche Segmenthoehenangabe in Pflanze Nr. :");
       WriteCard(Pflanzenzaehler,3); WriteLn;
       Ende:=TRUE;
      END; |
    9..MaxElement :
      CASE Elementzaehler OF
       9,13,17 :
       IF (IZahl>0) OR (IZahl<=MaxFarbenNr) THEN
        FarbenNr[Pflanzenteile((Elementzaehler-8) DIV 4)]:=IZahl;
        Temp:=Elementzaehler;
        INC(Elementzaehler);
       ELSE
        WriteString("Falsche Farbennummer in Pflanze Nr. :");
        WriteCard(Pflanzenzaehler,3); WriteLn;
        Ende:=TRUE;
       END;  |
      ELSE
       IF (IZahl>0) OR (IZahl<=MaxFarbenanteil) THEN
        Farbenanteil[Pflanzenteile((Temp-8) DIV 4),
               Grundfarben(Elementzaehler-Temp-1)]:=IZahl;
        INC(Elementzaehler);
       ELSE
        WriteString("Falsche Farbkomponentenangabe in Pflanze Nr. :");
        WriteCard(Pflanzenzaehler,3); WriteLn;
        Ende:=TRUE;
       END;
      END; |
     END;
    END;
    IF Ende OR (Elementzaehler > MaxElement) THEN Weiter:=TRUE; END;
    Zahl:=0.0;
   END;
  END;
 END;
END;
END;
END;
END;
 IF (Elementzaehler > 1) AND (Elementzaehler <= MaxElement) THEN
  WriteString("Nicht genug Angaben fueür Pflanze :");
  WriteCard(Pflanzenzaehler,3); WriteLn;
  Ende:=TRUE;
 ELSIF (Elementzaehler > MaxElement) THEN
  LeseVorgangOK:=TRUE;
  INC(Pflanzenzaehler);
 END;
RETURN LeseVorgangOK;
```

```
END LiesEinePflanze;


BEGIN
Lookup(INFile,Filename,0,FALSE);
IF INFile.res # done THEN
 WriteString("Fehler beim Oeffnen des Files ");
 WriteString(Filename); WriteLn;
 WriteString("Aufgetretener Dateifehler : ");
 ResponseText(INFile.res,StrPtrv);
 WriteString(StrPtrv^); WriteLn;
 Ende:=TRUE;
ELSE
 FileEnde:=FALSE;
 Pflanzenzaehler:=1;
 REPEAT
  IF LiesEinePflanze() THEN
   LSYSTEM(Pflanze.Wachstumsstufe,Saat,LKETTE);
   InitPflanzenFarben;
   OK:=ZeichnePflanze(0,0,0,Pflanze.Wurzelneigung);
  ELSE
   FileEnde:=TRUE;
  END;
 UNTIL FileEnde
END;
Close(INFile);
END PerformPflanzenFile;


BEGIN
IF NumArgs() # 1 THEN
 WriteString("usage : [pflanze Datendatei]"); WriteLn;
ELSE
Ende:=FALSE;
OPENALL;
INITALL;
GetArg(1,Filename,FilenameL);
PerformPflanzenFile(Filename);
MenuOK:=SetMenuStrip(w,ADR(MenuBalken));
IF NOT(Ende) THEN
REPEAT
 WaitPort(w^.userPort);
  LOOP
   MsgPtr:=GetMsg(w^.userPort);
   IF MsgPtr=NIL THEN EXIT END;
   class:=MsgPtr^.class; code:=MsgPtr^.code;
    ReplyMsg(MsgPtr);
    IF (class=IDCMPFlagSet{closeWindow}) THEN
     Ende:=TRUE;
     EXIT;
    ELSIF (class=IDCMPFlagSet{menuPick}) AND (code#menuNull) THEN
     IF HauptMenus(MenuNUM(code))=HauptMenus(pflanzenmenu) THEN
      CASE Pflanzenmenus(ItemNUM(code)) OF
         Pflanzenmenus(neuzeichnen) :
          ClearScreen(rp);
          PerformPflanzenFile(Filename);
          EXIT; |
         Pflanzenmenus(ende) :
          Ende:=TRUE;
          EXIT; |
      END;
     END;
    END;
  END;
UNTIL Ende;
END;
END;
GOODBYE;
END pflanze.
```

AMIGA-SOFTWARE: NUR VOM FEINSTEN



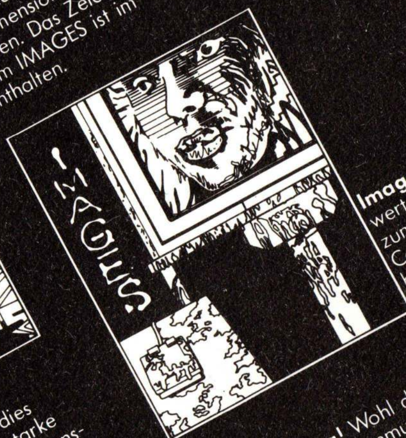
Neu! Videoscape 3 D deutsche PAL Version. Erstellen Sie Ihre eigenen Trickfilme, lassen Sie die Grafiken dreidimensional aus dem Bildschirm fliegen! Updatehandbuch und Diskette gegen Einsendung des engl. Handbuches und 49.– DM Schutzgebühr erhältlich.



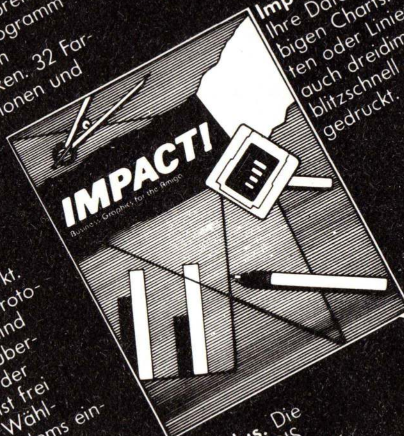
Neu! Videotitler, das Nonplusultra zur Erzeugung von Titelvorspännen und Präsentationen. Natürlich 3 D und beliebig mit anderen Grafiken mischbar. Extrem kurze Berechnungszeiten bei perspektivischer Darstellung.



Animator, die preiswerte Alternative zur Erstellung von zweidimensionalen Animationen. Das Zeichenprogramm IMAGES ist im Paket enthalten.



Images, ein sehr preiswertes Zeichenprogramm zum Erstellen von Computergrafiken. 32 Farben, Brushoptionen und vieles mehr.



Impact. Präsentieren Sie Ihre Daten mit eigenen farbigen Charts. Balken-, Torten- oder Liniendiagramme auch dreidimensional sind blitzschnell erstellt und ausgedruckt.



Sonix. Wer kennt dies nicht, das leistungsstarke vierstimmige Kompositionsprogramm für den AMIGA. Verarbeitet auch digitalisierte Instrumente oder Sprache und ist voll multitaskfähig.



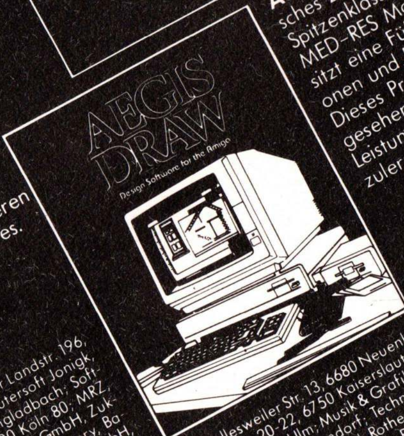
Diga! Wohl das ausgefeilteste Kommunikationsprogramm auf dem Markt. Alle nur erdenklichen Protokolle und Parameter sind während der Datenübertragung änderbar, der Protokollspeicher ist frei dimensionierbar. Wähloption für div. Modems eingebaut.

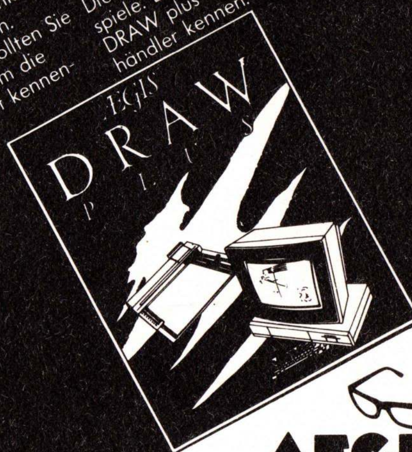
Aegis Draw plus. Die Steigerung von AEGIS DRAW. Arbeitet mit voller HIRES Auflösung und beherrscht alle Tricks. Ob automatisches Bemaßen, drehen, kopieren, kippen, vergrößern und verkleinern. Dies sind nur wenige Beispiele. Lernen Sie AEGIS DRAW plus bei Ihrem Fachhändler kennen.

Aegis Draw, ein technisches Zeichenprogramm der Spitzenklasse. Arbeitet im MED-RES Modus und besitzt eine Fülle von Funktionen und Parametern. Dieses Programm sollten Sie gesehen haben, um die Leistungsfähigkeit kennenzulernen.



Audiomaster. Digitalisieren und bearbeiten Sie nach Herzenslust Ihre eigenen Klänge und Sprache. Audiomaster arbeitet mit allen Digitizern zusammen und nutzt den vollen Rechnerspeicher. Variable Samplerate, Resampeln, Echo, Quasistereo, HIFI-Save, Mixen und Kopieren sind nur einige Features.





AEGIS Produkte und andere DTM AMIGA Software erhalten Sie u.a. bei: WAW Elektronic, Tegeler Str. 2, 1000 Berlin; Boysen & Maasch, Hermannstr. 31, 2 Hamburg; Hamburger Softwareladen, Gärtnerstr. 5, 2000 Hamburg; System Shop, Lilienstr. 23, 2000 Hamburg; Reese GmbH, Rendsburger Landstr. 196, 2300 Kiel; GOLD Computerhaus, Donnerschweerstr. 127, 2900 Oldenburg; CSJ Computersoft Jonigk, An der Tiefenriede 27, 3000 Hannover; Diamond Soft, Regentstr. 178, 4050 Mönchengladbach; Softshop, Müllersgasse 6-8, 4100 Duisburg 1; International Software, Heidenrichstr. 10, 5000 Köln 80; MRZ, Kölner Landstr. 240, 5160 Düren; Hansen & Gieraths EDV, Münsterstr. 1, 5300 Bonn; Kaurisch GmbH, Zukerbergstr., 5500 Trier; Mediencenter, Wermingser Str. 45, 5860 Iserlohn; Intelligent Memory, Basaltstr. 58, 6000 Frankfurt; Computer Landolt, Wingertstr. 114, 6457 Maintal 1; Kaurisch GmbH, Trierer Str. 16-22, 6600 Saarbrücken; SHOP 64, Großherzog Friedrich-Str. 40, 6600 Saarbrücken; SHOP 64, Titzstr. 24, 6630 Saarlouis; SHOP 64, Talstr. 44, 6650 Homburg; SHOP 64, Ludwigstr. 46, 6670 St. Ingbert; Proficenter, Lutherstr. 7, 6680 Neunkirchen; SHOP 64, Bellesweiler Str. 13, 6680 Neuenkirchen; Profisound, Schillerstr. 36, 6700 Ludwigshafen; Kaurisch GmbH, Richard Wagner Str. 20-22, 6750 Kaiserslautern; Gausch & Sturm, Casterfeldstr. 74, 6800 Mannheim; Computerstudio, Hasenbad 18/1, 7900 Ulm; Musik & Grafiksoftware, Wasserburger Landstr. 244, 8000 München; Technoland, Werftstr. 15, 8360 Deggendorf; Technoland, Neuburgerstr. 104, 8390 Passau; Technoland, Lilienthalstr. 3, 8400 Regensburg; Technoland, Rothenburger Str. 451, 8500 Nürnberg; Schöll Büroteam, Dominikanerplatz 5, 8700 Würzburg; Bissinger Bürotechnik, Industriestr. 18, 8883 Gundelfingen; Adolf Schmoll Computerstudio, Schwalbenstr. 1, 8900 Augsburg sowie bei **HAKO AG** Foto, Video, Elektronik, **SATURN HANSA** und andeen guten Fachhandelsgesellschaften. Vertriebsrechte für die Bundesrepublik Deutschland: **DTM Werbung und EDV GmbH,** Poststr. 25, 6200 Wiesbaden-Bierstadt, Telefon (06121) 560084, fax (06121) 563643

AEGIS DEVELOPMENT

MICROTRON
COMPUTERPRODUKTE
Postfach 69 Bahnhofstr. 2
CH-2542 PIETERLEN

DTM
Poststraße 25
6200 Wiesbaden-Bierstadt
(0 61 21) 56 00 84
fax (0 61 21) 56 36 43
Werbung und EDV GmbH



Impressum

KICKSTART Grafik Spezial

Chefredakteur:

Uwe Bärtels (Chefredakteur)
Markus Nerding (Stellvertreter)

Leitender Redakteur:

Wolf Dietrich

Redaktion:

Andreas Krämer
Gerald W. Carda
Harald Schneider
Marcelo Merino
Harald Egel

Herausgeber:

'MERLIN'-Computer GmbH
Industriestraße 26
Postfach 5569
6236 Eschborn
Tel.: 06196/481811
FAX: 06196/41137

Autoren dieser Ausgabe:

Andreas Diebold
Oliver Edler
Jobst Hermeyer
Christian Mönch
Frank Schäfer
Jürgen Schmidt
Christian Schormann
Michael Sistig

Public Relations:

Claus Peter Lippert

Auslandskorrespondenz:

D. dela Fuente (GB)
L. Hennely (USA)

Verlag:

Heim Verlag
Heidelberger Landstraße 194
6100 Darmstadt 13
Tel.:06151/56057
FAX:06151/55689
FAX:06151/56059

Verlagsleitung:

Hans-Jörg Heim

Anzeigenverkauf:

Kyriakulla Margaritis
Uwe Heim (Ltg.)

Anzeigenpreise:

nach Preisliste Nr.3, gültig ab 1.1.88

Grafische Gestaltung, DTP:

Fabian & Maier

Produktion:

Karl-Heinz Hoffmann
Klaus Schultheis
Susanne Failer
Bernd Failer

Fotographie:

Rainer Spirandelli, Archiv

Titelbild:

ArtCom, Bremen

Druck:

Ferling Druck, Darmstadt

Bezugsmöglichkeit:

Zeitschriftenhandel, Kauf- und Warenhäuser, Commodore-Fachhändler oder direkt beim Verlag.

Einzelpreis: DM 14,- ,ÖS 112,- SFr 14,-

GRAFIK SPEZIAL

PROGRAMMDISKETTE

Alle in diesem Sonderheft veröffentlichten Programme inklusive der dazugehörigen Quelltexte sind auch auf Diskette erhältlich.

Diese bietet sich für alle Anwender an, die die Programme nutzen, aber die Listings nicht abtippen möchten, oder für diejenigen, die nicht über den jeweils verwendeten Compiler (C oder Modula 2) verfügen.

Die Diskette (nebenstehend abgebildet) enthält folgende Programme:

- Programm Picswitch inkl. C-Source

Dieses Programm dient zum Erstellen von Animationen aus IFF- Bildern, wozu ein Differenzbildverfahren verwendet wird, das die Größe der Animationsdateien reduziert und die Abspielgeschwindigkeit erhöht.

- Programm Transformer inkl. C-Source

Transformer arbeitet nach dem Prinzip des Texture Mapping, womit Sie beliebige IFF-Bilder um Kugeln oder Röhren gewickelt darstellen können.

- Programm Pflanze inkl. Modula 2-Source

Dieses Programm simuliert mathematisch natürliche Wachstumsformen und stellt diese grafisch dar.

- Programm WBCop inkl. Modula 2-Source

Ein Programm zum Verändern der Farblisten eines Workbenchbildschirms. Es können sowohl der Hintergrund wie auch der Text oder die Umrandungsfarben in durchlaufenden Farben dargestellt werden.

- Programm ObjConvert in BASIC

Dieses Programm dient der Übertragung von Objektdateien von Sculpt 3D zu Videoscape 3D.

Desweiteren finden Sie alle Listings und die dazugehörigen Programme der Grundlagenartikel dieses Hefts auf dieser Diskette.



Bitte senden Sie mir die Original-Programmdiskette zur KICKSTART GRAFIK SPEZIAL zum Preis von DM 25.-

Ich zahle keine Versandkosten
Den Betrag begleiche ich durch beigefügten Verrechnungscheck

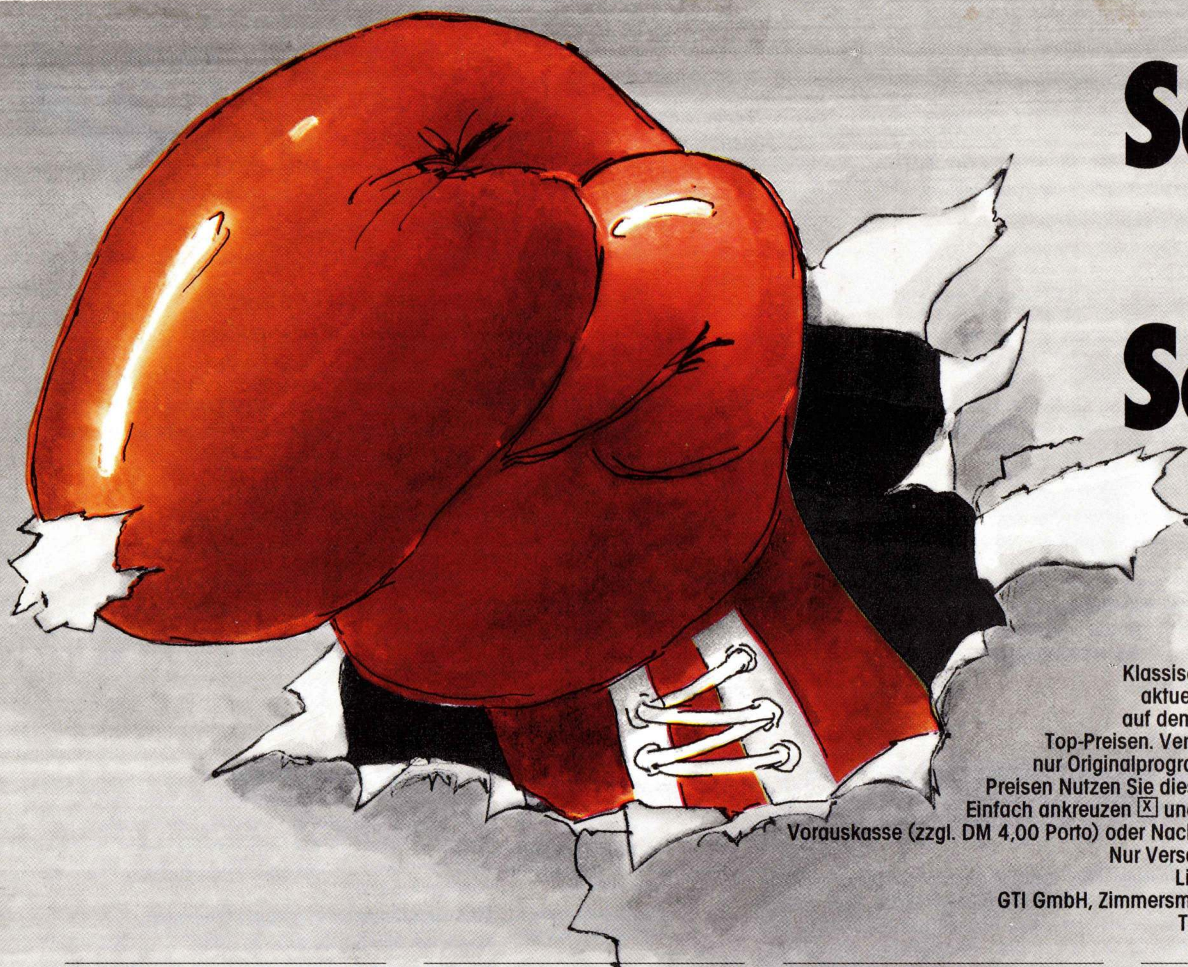
Name:____________________

Straße:____________________

Plz/Ort:____________________

Einsenden an:

Heim-Verlag
Heidelberger Landstraße 194
6100 Darmstadt/Eberstadt



Schlag auf Schlag

Software für Ihren AMIGA

Klassische Renner ebenso wie brand-aktuelle Programme, z. Tl. erstmalig auf dem deutschen Markt. Top-Hits zu Top-Preisen. Vergleichen Sie selbst! Wir liefern nur Originalprogramme zu knallhart kalkulierten Preisen Nutzen Sie diese Anzeige für Ihre Bestellung: Einfach ankreuzen ☒ und ab zur Post! Lieferung gegen Vorauskasse (zzgl. DM 4,00 Porto) oder Nachnahme (zzgl. DM 6,00 Porto). Nur Versandhandel, kein Ladenverkauf. Lieferung, solange Vorrat reicht.

GTI GmbH, Zimmersmühlenweg 73, 6370 Oberursel
Telefon (0 61 71) 5 38 63/37 48

Programm	DM
SPIELE UND SIMULTATIONEN	
☐ Adventure Construction Set	49,95
☐ Alien Fires	69,00
☐ Alien Strike	49.95
☐ Arazok's Tomb	65,00
☐ Arkanoid	69,00
☐ Backlash	54,90
☐ Bad Cat	59,00
☐ Balance of Power	95,00
☐ Barbarian	74,95
☐ Bards Tale	89,00
☐ Breach	74,95
☐ Bureaucracy	75,00
☐ Championship Golf	75,00
☐ City Defence	29,00
☐ Cogans Run	49,00
☐ Crazy Cars	64,90
☐ Dark Castle	75,00
☐ Defender of the Crown	85,00
☐ Deja Vu	85,00
☐ Destroyer	74,95
☐ Diablo	54,95
☐ Emerald Mine	29,00
☐ Emetic Skimmer	49,00
☐ Faery Tale Adventure	85,00
☐ Feud	29,00
☐ Firepower	49,95
☐ Flight Simulator II	99,00
☐ F.Sim Scenery Disk # 7	49,90
☐ F.Sim Scenery Disk # 11	49,90
☐ Footman	49,95
☐ Fortress Underground	29,00
☐ Fußball Manager	74,95
☐ Galactic Invasion	44,95
☐ Galaxy Fight	39,00
☐ Galileo	105,00
☐ Garrison	69,00
☐ Girls of Riviera	44,95
☐ Goldrunner	75,00
☐ Gnome Ranger	44,95
☐ Guild of Thieves	64,90
☐ Halley Project	85,00
☐ Hollywood Hijinx	75,00
☐ Hollywood Poker	39,00
☐ Indoor Sports	90,00
☐ Insanity Fight	75,00
☐ Into the Eagles Nest	75,00
☐ Impact	49,95
☐ Jagd auf Roter Oktober	74,95
☐ Jet	89,95
☐ Jinxter	79,00

Programm	DM
☐ Karate Kid II	69,95
☐ Karting Grand Prix	29,00
☐ Knight Orc	59,00
☐ Land of Lounge Lizards	90,00
☐ Leviathan	69,00
☐ Love Quest	110,00
☐ Mercenary	64,90
☐ Mike the Magic Dragon	29,00
☐ Mission Elevator	59,95
☐ Moebius	79,95
☐ Mouse Trap	44,95
☐ Phalanx II	29,00
☐ Phantasie III	59,95
☐ Pinnball Wizard	49,00
☐ Plutos	49,95
☐ Q-Ball	59,00
☐ Roadwars	54,90
☐ Roadwar 2000	59,95
☐ Roadwar Europa	59,95
☐ Shadowgate	89,00
☐ Silicon Dreams	59,00
☐ Sinbad	85,00
☐ Space Port	54,95
☐ Space Quest	89,00
☐ Space Ranger	29,00
☐ Speed	29,00
☐ Starfleet I	99,00
☐ Star Glider	69,00
☐ Super Huey	60,00
☐ Swooper	59,95
☐ Terrorpods	74,95
☐ Test Drive	74,95
☐ The Final Trip	29,00
☐ The Wall	49,00
☐ Thunderboy	59,00
☐ Time Bandit	54,90
☐ Tolteka	59,00
☐ Uninvited	85,00
☐ Vyper	49,95
☐ Western Games	59,00
☐ Winter Games	75,00
☐ Winter Olympiad 88	54,95
☐ World Games	75,00
☐ Xenon	54,90
☐	
☐ XR-35	29,00
SCHACHECKE	
☐ Art of Chess	64,90
☐ Chessmaster 2000	80,00
☐ Großmeister	74,95
☐ Sargon III	95,00

Programm	DM
ANIMATIONS- UND GRAFIKSOFTWARE TEXTVERARBEITUNG UND DESKTOP PUBLISHING	
☐ Aegis Animator/Images	235,00
☐ Aegis Draw Plus	445,00
☐ Aegis Video Titler	249,00
☐ Analytic Art	110,00
☐ Animate 3D	275,00
☐ Animator Flipper	70,00
☐ Animator Junior	135,00
☐ Butcher (Deutsch PAL)	110,00
☐ Deluxe Paint II (Deutsch PAL)	249,00
☐ Desktop Artist	55,00
☐ Digipaint (Deutsch PAL)	138,00
☐ Digipic	1100,00
☐ Digiview (Deutsch PAL)	440,00
☐ Digiview PAL Upgrade	19,90
☐ Director	125,00
☐ Express Paint	135,00
☐ Forms in Flight	145,00
☐ Gender Changer	55,00
☐ Graphic Studio	105,00
☐ Interchange	85,00
☐ Page Flipper	85,00
☐ PIX mate	120,00
☐ Prism +	120,00
☐ Professional Page	660,00
☐ Sculpt 3D	190,00
☐ Silver 3D	280,00
☐ TV Show	185,00
☐ TV Text	179,00
☐ Word Perfect Installer	69,00
☐ X-CAD Designer	1145,00
☐ Zuma Fonts Volume I	60,00
☐ Zuma Fonts Volume II	60,00
☐ Zuma Fonts Volume III	60,00
PROGRAMMIERSPRACHEN UND UTILITIES	
☐ AC Basic	360,00
☐ AC Fortran	545,00
☐ Aztec C 3.6 (DEV)	595,00
☐ Go – 64	139,90
☐ Grabbit	54,00
☐ Intswitch	27,50
☐ Lattice C 4.0	385,00
☐ LV Backup	120,00
☐ M2 Amiga (Deutsch)	350,00
☐ Marauder II	69,00
☐ Metacomco Assembler	185,00
☐ Metacomco Pascal	185,00
☐ Metacomco Shell	135,00
☐ Mimic	69,00
☐ Mirror Copier	85,00

Programm	DM
☐ Modula 2 (Regular)	185,00
☐ Modula 2 (Developers)	275,00
☐ Modula 2 (Commercial)	545,00
☐ Quarterback	135,00
☐ The 64 Emulator	148,00
☐ True Basic	195,00
☐ Turbo Print	89,00
MUSIKPROGRAMME	
☐ Aegis Audiomaster	99,00
☐ Aegis Sonix	135,00
☐ Casio CZ Editor/Librarian	225,00
☐ Deluxe Music Construction Set	199,00
☐ Dynamic Drums	135,00
☐ Dynamic Studio	375,00
☐ DX7 Master Editor/Librarian	275,00
☐ D50 Master Editor/Librarian	275,00
☐ ECE MIDI Interface	130,00
☐ ECT Sample Ware	145,00
☐ Generic Editor/Librarian	225,00
☐ Instant Music	85,00
☐ Pro MIDI Studio	345,00
☐ Soundsampler	225,00
DATENFERNÜBERTRAGUNG	
☐ Aegis Diga	135,00
BUSINESSPROGRAMME	
☐ Acquisition 1.3F	545,00
☐ Aegis Impact	150,00
☐ Haicalc	105,00
☐ Logistix (Deutsch)	340,00
☐ Maxiplan 500	249,00
☐ Maxiplan Plus	345,00
☐ Microfiche Filer	175,00
BÜCHER	
☐ Balance of Power Book	24,95
☐ Music Through MIDI	39,95
☐ Robo City News	4,95
VERSCHIEDENES	
☐ Flicker Master	35,00
☐ Lipstick Plus	44,95
☐ Megacover (500 + Maus)	29,95
☐ Mouse House	19,90
☐ Mouse Mat	16,50

GTI. Spezialist für AMIGA-Software

